



INGENIERÍA TÉCNICA INDUSTRIAL:
ELECTRÓNICA INDUSTRIAL

PROYECTO FIN DE CARRERA:

**DISEÑO Y DESARROLLO DE UNA INTERFAZ GRÁFICA DE USUARIO PARA
LA PRUEBA DE DAQS BASADOS EN ARDUINO MEDIANTE ROS.**

AUTOR: DANIEL MARTIN DE CONSUEGRA MARTINEZ

TUTORES: RAMÓN BARBER CASTAÑO

DAVID GARCIA GODOY

NOVIEMBRE 2012



AGRADECIMIENTOS

Quiero agradecer en primer lugar a mi familia y a mi novia, las personas mas cercanas y las que más me han ayudado.

A mis compañeros de piso en particular por sus consejos y ayudas y mis amigos por su comprensión

A la universidad por estos años de carrera inolvidables y a mi tutores por la contribución a este proyecto que culmina una fase de mi vida.

Contenido

AGRADECIMIENTOS.....	3
Capítulo 1: INTRODUCCIÓN.....	11
1.1.Motivación.	11
1.2. Objetivos.	12
1.3.Partes del documento.	14
Capítulo 2: ARDUINO.....	15
2.1. Introducción	15
2.2. Tipos	16
2.3. Arduino Mega 2560.....	20
2.3.1. Introducción	20
2.3.2. Características	21
2.3.3. Alimentación de la placa	21
2.3.4. Pines de la placa	22
2.3.7. Programación	25
2.4.1. Librerías de Arduino	28
Capítulo 3: QT.....	29
3.1. Introducción	29
3.2. Plataformas	30
3.3. Bindings	30
3.4. QT Creator	31
3.4.1. Introducción	31
3.4.2. Características	33
3.4.3. Sistemas operativos soportados	33
3.4.4. Gestión de proyectos	34
3.4.5. Diseño de interfaces de usuario.....	36
3.4.6. Codificación	36
3.4.7. Creación y ejecución	38
3.5. Obtención de ayuda	39
3.6. Usando el compilador Mega-Object (MOC).....	40



3.6.1. Introducción	40
3.6.2. Uso.....	40
3.6.3. Escribir un reglamento para invocar MOC	41
3.6.4. Diagnóstico.....	41
3.6.5. Limitaciones	41
Capítulo 4: ROS.....	43
4.1. Introducción	43
4.2. Objetivos del diseño.....	44
4.3. Sistemas operativos	45
4.4. Conceptos.....	45
4.4.1. ROS nivel del sistema de archivos	45
4.5.1. Bibliotecas principales de clientes	48
4.5.2. Bibliotecas experimentales de clientes.....	49
4.7. Conclusión	50
Capítulo 5: IMPLEMENTACIÓN Arduino-ROS	51
5.1. Introducción	51
5.2. Librería Rosserial	51
5.3. Agregar mensajes personalizados.....	53
5.4. Descripción de un programa de Arduino	53
5.5. Flujograma.....	54
Capítulo 6: IMPLEMENTACIÓN ROS-QT	59
6.1 Creación proyecto	59
6.1.1. Creación paquete ROS.....	59
6.1.2 Creación mensajes personalizados ROS.....	59
6.1.3. Modificación cMakelist.txt.....	60
6.2. Flujograma principal.....	62
6.3. Clase principal Arduino	63
6.3.1. Flujograma Arduino.....	63
6.3.2 VOID CREAR ANALOGICO ():.....	64
6.3.3. VOID CREAR PWM ():	66
6.3.4. VOID CREAR DIGITAL ():.....	67
6.3.1 PIN	69
6.3.2 PWM.....	70



6.3.3 NODO	71
6.4. Implementación funcionalidad: SIGNALS Y SLOT	72
Capítulo 7: RESULTADOS EXPERIMENTALES	75
7.1. Introducción.	75
7.2. Inicialización.	75
7.2.1. Ejecución SERIALNODE	76
7.2.2. Ejecución INTERFAZ_ARDUINO	77
7.3. Salidas digitales.	78
7.3.1. Primer caso.....	78
7.3.2. Segundo caso.....	80
7.4. Entradas digitales.	81
7.5. Entradas analógicas.....	84
7.5.1. Distancia larga	85
7.5.2. Distancia media	87
7.5.3. Distancia corta.....	88
7.5.4. Fluctuaciones.....	89
7.6. Salidas PWM.....	90
7.6.1. Casos analizados.....	90
7.7. Mediciones varias PWM.....	95
Capítulo 8: CONCLUSIONES Y TRABAJOS FUTUROS.	97
8.1. Conclusiones.....	97
8.2. Trabajos futuros	98
Referencias.....	100
ANEXOS.	101
ANEXO 1. Ros/qt.....	102
A1.1.MAIN.CPP	102
A.1.2. ARDUINO.H	103
A1.3. ARDUINO.CPP	105
A.1.4. PIN.H	110
A.1.5. PIN.CPP.....	111
A 1.6. PWM.H	112
A1.7. PWM.CPP	112
A1.8. NODO.H.....	113



A1.9. NODO.CPP	114
ANEXO 2 arduino/ros	116

Índice de figuras

FIGURA 1. 1. FUNCIONAMIENTO GENERAL	12
FIGURA 1. 2. COMUNICACIÓN ROS	13
FIGURA 2. 1. ARDUINO UNO	16
FIGURA 2. 2. ARDUINO LILYPAD	17
FIGURA 2. 3. ARDUINO MEGA.....	17
FIGURA 2. 4. ARDUINO FIO	17
FIGURA 2. 5. ARDUINO PRO.....	18
FIGURA 2. 6. ARDUINO NANO	18
FIGURA 2. 7. ARDUINO MINI.....	18
FIGURA 2. 8. ARDUINO PRO MINI	19
FIGURA 2. 9. USB/SERIAL LIGHT ADAPTER	19
FIGURA 2. 10. ARDUINO MEGA 2560	20
FIGURA 2. 11. ALIMENTACIÓN DE ARDUINO.....	21
FIGURA 2. 12. PINES DE ALIMENTACIÓN	22
FIGURA 2. 13. PINES DIGITALES	22
FIGURA 2. 14. PINES PWM.....	23
FIGURA 2. 15. PINES ANALÓGICOS.....	23
FIGURA 2. 16. PIN AREF.....	24
FIGURA 2. 17. PIN RESET	24
FIGURA 2. 18. ENTORNO ARDUINO	25
FIGURA 2. 19. SELECCIÓN DE LA PLACA.....	26
FIGURA 2. 20. SELECCIÓN DE PUERTO EN WINDOWS	26
FIGURA 2. 21. SELECCIÓN DE PUERTO EN LINUX	27
FIGURA 2. 22. BARRA DE HERRAMIENTAS	27
FIGURA 2. 23. SELECCIÓN DE LIBRERÍAS.....	28
FIGURA 3. 1. QT CREATOR[5]	31
FIGURA 3. 2. NUEVA QT GUI PROYECTO APLICACIÓN ASISTENTE	34
FIGURA 3. 3. EDITOR DE CÓDIGO EN MODO DE EDICIÓN.....	36
FIGURA 3. 4. EL MISMO CÓDIGO FUENTE CONSTRUIDO SE EJECUTA EN MÚLTIPLES OBJETIVOS	38
FIGURA 3. 5 CONTEXTO SENSIBLE DE AYUDA QT.....	39
FIGURA 4. 1 COMPUTACIÓN A NIVEL GRÁFICO ROS	47
FIGURA 5. 1. LIBRERÍA ROS	52
FIGURA 5. 2. FLUJOGRAMA IDE-ARDUINO	54
FIGURA 5. 3. FUNCIÓN PIN	56
FIGURA 5. 4. FUNCIÓN PWM	57
FIGURA 6. 1. INTERACCIÓN DE ARCHIVOS	62
FIGURA 6. 2. FLUJOGRAMA CLASE PRINCIPAL	63
FIGURA 6. 3. FLUJOGRAMA DE LA CREACIÓN Y COLOCACIÓN DE ENTRADAS ANALOGICAS.....	64
FIGURA 6. 4. REPRESENTACIÓN DE LAS ENTRADAS ANALÓGICAS.....	65
FIGURA 6. 5. FLUJOGRAMA DE LA CREACIÓN PWM.....	66

FIGURA 6. 6. REPRESENTACIÓN DE LAS PWM	67
FIGURA 6. 7. FLUJOGRAMA DE LA FUNCIÓN CREAR_DIGITAL	67
FIGURA 6. 8. ENTRADAS DIGITALES	68
FIGURA 6. 9. SALIDAS DIGITALES	68
FIGURA 6. 10. FLUJOGRAMA PIN.....	69
FIGURA 6. 11. FLUJOGRAMA PWM	70
FIGURA 6. 12. FLUJOGRAMA CLASE NODO.....	71
FIGURA 6. 13. FUNCIONALIDAD PIN.....	72
FIGURA 6. 14. FUNCIONALIDAD ANALÓGICO.....	73
FIGURA 6. 15. FUNCIONALIDAD PWM	74
FIGURA 7. 1. INICIALIZACIÓN ROSCORE	75
FIGURA 7. 2. INICIALIZACIÓN SERIAL/NODO	76
FIGURA 7. 3. INICIALIZACIÓN DE INTERFAZ.....	77
FIGURA 7. 4. APLICACIÓN INTERFAZ_ARDUINO	77
FIGURA 7. 5. PRIMER MONTAJE DE ENTRADAS DIGITALES	78
FIGURA 7. 6. PANEL SALIDAS DIGITALES	79
FIGURA 7. 7. SEGUNDO MONTAJE DE ENTRADAS DIGITALES	80
FIGURA 7. 8. PANEL SALIDAS DIGITALES	81
FIGURA 7. 9. INTERFAZ SALIDAS A ANALÓGICAS A 5V.....	82
FIGURA 7. 10. INTERFAZ SALIDAS A ANALÓGICAS A 0V.....	82
FIGURA 7. 11. INTERFAZ SALIDAS A ANALÓGICAS A 0-5V	83
FIGURA 7. 12. CAMBIOS DE ESTADO PIN_DIGITAL.....	83
FIGURA 7. 13. CAMBIOS DE ESTADO MÁS VELOCES	84
FIGURA 7. 14. MONTAJE SENSOR.....	85
FIGURA 7. 15. SENSOR DISTANCIA LARGA.....	86
FIGURA 7. 16. DETALLE DISTANCIA LARGA.....	86
FIGURA 7. 17. SENSOR DISTANCIA MEDIA	87
FIGURA 7. 18. DETALLE DISTANCIA MEDIA	87
FIGURA 7. 19. SENSOR DISTANCIA CORTA	88
FIGURA 7. 20. DETALLE DISTANCIA CORTA	88
FIGURA 7. 21. FLUCTUACIÓN MEDIA CORTA	89
FIGURA 7. 22. FLUCTUACIONES RÁPIDAS	89
FIGURA 7. 23. CICLO DE TRABAJO AL 10%	90
FIGURA 7. 24. CICLO DE TRABAJO AL 25%	91
FIGURA 7. 25. CICLO DE TRABAJO AL 50%	91
FIGURA 7. 26. DETALLE DEL CASO AL 50%.....	92
FIGURA 7. 27. CICLO DE TRABAJO AL 75%	92
FIGURA 7. 28. DETALLE DEL CASO AL 50%.....	93
FIGURA 7. 29. CICLO DE TRABAJO AL 95%	93
FIGURA 7. 30. DETALLE DEL CASO AL 95%.....	94
FIGURA 7. 31. MÚLTIPLES LED'S	95
FIGURA 7. 32. INTERFAZ PARA CONTROLAR MÚLTIPLES LED'S.....	95



Capítulo 1: INTRODUCCIÓN

1.1.Motivación.

La robótica es el arte de percibir y manipular a través de dispositivos controlados por computador, el mundo real. Los robots son capaces de captar datos del mundo a través de diferentes sensores (láser, sonar) y de ejecutar diferentes operaciones gracias a actuadores. Sus aplicaciones son muy diversas, existen dispositivos móviles dedicados a la exploración de planetas, robots diseñados para usos industriales, robots de utilización en cirugía asistida, etc.

Un microcontrolador es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. Está compuesto de varios bloques funcionales, los cuales cumplen una tarea específica. Incluyen en su interior las tres principales unidades funcionales de una computadora: unidad central de procesamiento, memoria y periféricos de entrada/salida. Es por ello por lo que se requieren en el mundo de la robótica como elementos de control y procesamiento de los robots debido a su pequeño tamaño y prestaciones. Estos son los encargados de controlar los sensores que reciben la información del medio mediante sus entradas y controlan la funcionalidad a través de sus salidas (servomotores, iluminación...).

Arduino aparece como uno de los micros más usados por su sencillez, su facilidad de programación, el número de librerías disponibles, y su bajo coste. Es por ello por lo que se ha elegido para este proyecto pensando en su integración con la robótica. Por ello surge la necesidad de enmarcarlo en un software diseñado para la robótica como es ROS y la creación de una interfaz que permita de forma sencilla la manipulación y lectura de sus entradas y salidas mediante un PC.

1.2. Objetivos.

En este proyecto se busca la implementación de un Arduino como tarjeta de adquisición de datos, utilizando para el flujo de control de datos el software de ROS y la creación de una simple interfaz de usuario con QT para la visualización y manipulación de estos datos.

Este objetivo puede desglosarse:

- Integración del Arduino con el software Ros a través de su librería rosserial
- Integración del código de QT 4 Creator en un paquete de Ros, mediante la implementación de su CMakeList.
- Implementación del código necesario para la lectura y escritura de las E/S digitales y analógicas mediante la suscripción y publicación de topics en ROS
- Implementación del código necesario para la visualización y manipulación de datos en una interfaz de usuario creada en código de QT 4.

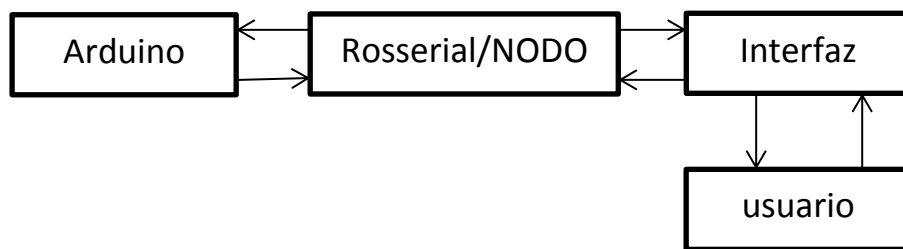


Figura 1. 1. Funcionamiento general

Para la consecución de estos objetivos se ha realizado:

- El estudio de Arduino y de su entorno de programación, así como del funcionamiento de sus pines y puertos serie y el encapsulado en librerías de su código.
- Un estudio del funcionamiento general de ROS, configuración de variables y topics a través de la publicación y suscripción de mensajes.
- Estudio del entorno de desarrollo Qt, codificación, tipo de widgets, signals y slots predefinidos y su posterior integración en un paquete ROS.

El primer paso fue la integración de ROS y Arduino mediante la librería predeterminada de Ros rosserial, este software permite el envío de mensajes en Ros, publicados en diferentes temas llamados topics para el envío de estos datos se crearon mensajes específicos y topic en el IDE Arduino. En este paso se consiguió la comunicación completa de Arduino y ROS mediante el nodo serial node (implementado en rosserial).

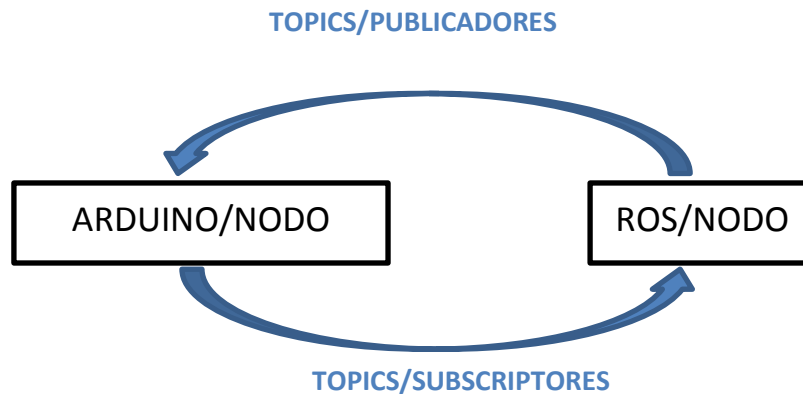


Figura 1. 2. Comunicación ROS

El código desarrollado en QT establece una interfaz de usuario para mostrar y manipular el flujo de datos. La Interfaz diseñada se ha implementado de forma simple, es decir se han colocado intuitivamente los botones y demás actuadores siguiendo el modelo de la placa de Arduino. Este código fue implementado junto con el código ROS en un paquete de éste, generando la aplicación QT y un nuevo Nodo para la comunicación con Arduino.

- ADQUISICION DE DATOS

La adquisición de datos o adquisición de señales, consiste en la toma de muestras del mundo real (sistema analógico) para generar datos que puedan ser manipulados por un ordenador u otros sistemas digitales. Consiste, en tomar un conjunto de señales físicas, convertirlas en tensiones eléctricas y digitalizarlas de manera que puedan ser procesadas. El elemento que hace dicha transformación es el módulo de digitalización o tarjeta de Adquisición de Datos **DAQ**. Se ha realizado un estudio de Arduino como tarjeta de adquisición de datos para la integración en la robótica y en concreto con ROS no exigen de unas altas prestaciones como DAQ.

La capacidad de Arduino como tarjeta de adquisición de datos depende de su frecuencia de muestreo, resolución y capacidad de respuesta:

- En primer lugar en el tema de la resolución el Arduino posee 10 bit de resolución. Los bits de resolución significan 2^{10} divisiones, o 1024 (0 a 1023), de la tensión de referencia. Para un Arduino, la tensión de referencia es de 5 voltios, y eso significa que la más pequeña variación de tensión detectable es $5/1023$ o 0,0049 voltios (4,9), la cual se trata de una precisión media. Los puertos Arduino ADC están normalmente atados a una referencia de 5 voltios, pero esta es ajustable. Es fácil utilizar el software para cambiar la tensión de referencia a 3,3 V que proporciona uno de los pines de Arduino, eso nos da una resolución de $3.3/1023$ o 0,0011. Específicamente el pin el AREF podría ser ajustado utilizando una fuente de tensión lo que permitirá obtener resoluciones más precisas si así lo requerimos.

- Frente a frecuencias de muestreo, el Arduino no se muestra preciso a menos que le obliguemos a ello por la tanto la manera más eficiente de calcular la frecuencia de muestreo es determinar una marca de tiempo y recoger su voltaje. Típicamente ofrecería un numero de entre 20 y 30 muestras por segundo que resultaría útil para aplicaciones que no sean restrictivas en tiempo, sin en cambio podemos mejorar estas lecturas almacenando los datos en la memoria interna del Arduino pero ésta es sólo de 8K por lo que no sería adecuado para lecturas continuas.
- La capacidad de respuesta por otro lado no es un problema para Arduino pudiéndose obtener las muestras con suficiente calidad a una frecuencia de muestreo dada de hasta 8KHZ.

1.3.Partes del documento.

En el **Capítulo 1** se describen las motivaciones y objetivos del proyecto, se trata de un pequeño resumen de lo que se ha hecho y como se ha hecho.

En los **Capítulo 2, 3 y 4** se describe los elementos utilizados para el proyecto tanto el hardware utilizado como es Arduino (capitulo 3), con una descripción general de las placas Arduino y en concreto de la utilizada la mega2560. En los capítulos 3 y 4 se describe el software utilizado en el proyecto, con una descripción general de su funcionamiento posibilidades y utilidad, estos son ROS Y QT.

En el **Capítulo 5** se describe la implementación de Arduino y ROS: funcionalidad, librerías utilizadas, acoplamiento de estas, flujogramas y funcionalidad.

En el **Capítulo 6** se describe la implementación de ROS Y QT: cómo se lleva a cabo, archivos creados y su relación en la ejecución de la aplicación, flujogramas, su funcionalidad y cómo es conseguida.

En el **Capítulo 7** se detallan las prueban efectuadas, resultados obtenidos, circuitos propuestos y análisis de los resultados.

Por ultimo en el **Capítulo 8** se llegan a las conclusiones obtenidas y a futuros trabajos en pos de mejorar la eficiencia y funcionalidad del proyecto.

Capítulo 2: ARDUINO

2.1. Introducción

Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinarios.

El hardware consiste en una placa con un microcontrolador *Atmel AVR* y puertos de entrada/salida. Los microcontroladores más usados son el *Atmega168*, *Atmega328*, *Atmega1280*, *ATmega8* por su sencillez y bajo coste que permiten el desarrollo de múltiples diseños. Por otro lado el software consiste en un entorno de desarrollo que implementa el lenguaje de programación Processing/Wiring y el cargador de arranque (*boot loader*) que corre en la placa.

Arduino se puede utilizar para desarrollar objetos interactivos autónomos o puede ser conectado a software del ordenador (por ejemplo: Macromedia Flash, Processing, Max/MSP, Pure Data). Las placas se pueden montar a mano o adquirirse.

El entorno de desarrollo integrado libre se puede descargar gratuitamente [1] Al ser open-hardware, tanto su diseño como su distribución es libre, es decir, puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia.

Arduino, además de simplificar el proceso de trabajar con microcontroladores, ofrece algunas ventajas respecto a otros sistemas:

- **Asequible:** Las placas Arduino son más asequibles comparadas con otras plataformas de microcontroladores. La versión más cara de un módulo de Arduino puede ser montada a mano, e incluso ya montada cuesta bastante menos de 60€.

- **Multiplataforma:** El software de Arduino funciona en los sistemas operativos Windows, Macintosh OSX y Linux mientras que la mayoría de los entornos para microcontroladores están limitados a Windows.

- **Entorno de programación simple y directo:** El entorno de programación de Arduino es fácil de usar para principiantes y lo suficientemente flexible para los usuarios avanzados. Arduino está basado en el entorno de programación de Processing.

- **Software ampliable y de código abierto:** El software Arduino está publicado bajo una licencia libre y preparado para ser ampliado por programadores experimentados. El lenguaje puede ampliarse a través de librerías de C++, y si se está interesado en profundizar en los detalles técnicos, se puede dar el salto a la programación en el lenguaje AVR C en el que está basado.

- **Hardware ampliable y de Código abierto:** Arduino está basado en los microcontroladores *ATMEGA168*, *ATMEGA328* y *ATMEGA1280*. Los planos de los módulos están publicados bajo licencia “Creative Commons”, por lo que diseñadores de circuitos con experiencia pueden hacer su propia versión del módulo, ampliándolo u optimizándolo. Incluso usuarios relativamente inexpertos pueden construir la versión para placa de desarrollo para entender cómo funciona y ahorrar en costes.

Arduino tiene una página web[2] en la que se puede encontrar todo lo necesario para programar, desde la opción de descarga del entorno de desarrollo con sus instrucciones de instalación hasta una pequeña guía de programación. También se pueden encontrar ejemplos de programas y librerías que se pueden descargar. Además, cuenta con un foro en el que puedes exponer tus dudas y actualmente se está ampliando al español[3].

2.2. Tipos

En la página web [2] también se pueden encontrar los distribuidores a través de los cuales comprar las placas Arduino o los componentes necesarios para ser hechas a mano. Existe una gran variedad de placas Arduino ya montadas de fábrica, que depende del tamaño, número de pines o microcontrolador incorporado. De un mismo modelo existen varias versiones, ya que se van actualizando haciendo mejoras.



Figura 2. 1. Arduino Uno

Arduino Uno (Ver Figura 2.1.) es una placa electrónica basada en el ATmega328. Cuenta con 14 entradas / salidas digitales pines (de las cuales 6 se puede utilizar como salidas PWM), 6 entradas analógicas, un oscilador de cristal de 16 MHz, una conexión USB, un conector de alimentación, una cabecera ICSP, y un botón de reset. Contiene todo lo necesario para apoyar el microcontrolador, basta con conectarlo a un ordenador con un cable USB o con un adaptador de CA a CC o batería para empezar.

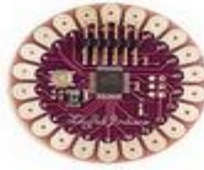


Figura 2. 2. Arduino LilyPad

El Arduino LilyPad (Ver Figura 2.2.) es una placa con microcontrolador diseñado para prendas y textiles. Puede utilizar complementos similares como fuentes de alimentación, sensores actuadores unidos por hilo conductor. La placa está basada en el ATmega168V (la versión de bajo consumo del ATmega168), o el ATmega328V (datasheet).



Figura 2. 3. Arduino Mega

El Arduino Mega (ver Figura 2.3.) es una placa microcontrolador basada ATmega1280 (datasheet). Tiene 54 entradas/salidas digitales (de las cuales 14 proporcionan salida PWM), 16 entradas digitales, 4 UARTS (puertos serie por hardware), un cristal oscilador de 16MHz, conexión USB, entrada de corriente, conector ICSP y botón de reset. Contiene todo lo necesario para hacer funcionar el microcontrolador; simplemente hay que conectar al ordenador con el cable USB o alimentarlo con un transformador o batería para empezar.



Figura 2. 4. Arduino Fio

El Arduino Fio(ver Figura 2.4.) es una placa microcontrolador basada en el ATmega328P (hoja de información). Funciona a 3.3V y 8MHz. Tiene 14 pines de E/S digitales (de los cuales 6 pueden usarse como salidas PWM), 8 entradas analógicas, un resonador en placa, un botón de reinicio (reset), y agujeros para montar conectores de pines. Tiene conexiones para una batería de polímero de Litio e incluye un circuito de carga a través de USB.



Figura 2. 5. Arduino Pro

La Arduino pro (ver Figura 2.5.) es una placa con un microcontrolador ATmega168 (datasheet) o en el ATmega328 (datasheet). La Pro viene en versiones de 3.3v / 8 MHz y 5v / 16 MHz. Tiene 14 E/S digitales (6 de las cuales se puedes utilizar como salidas PWM), 6 entradas analógicas, un resonador interno, botón de reseteo y agujeros para el montaje de tiras de pines. Vienen equipada con 6 pines para la conexión a un cable FTDI o a una placa adaptadora de la casa Sparkfun para dotarla de comunicación USB y alimentación.



Figura 2. 6. Arduino Nano

El Arduino Nano (ver Figura 2.6.) es una pequeña y completa placa basada en el ATmega328 (Arduino Nano 3.0) o ATmega168 (Arduino Nano 2.x) que se usa conectándola a una protoboard. No posee conector para alimentación externa, y funciona con un cable USB Mini-B en vez del cable estándar.



Figura 2. 7. Arduino Mini

Arduino Mini (ver Figura 2.7.) es una placa con un pequeño microcontrolador basada en el ATmega168 (datasheet), pensada para ser utilizada en placas de prototipo y donde el espacio es un bien escaso. Cuenta con 14 entradas/salidas digitales (de las cuales 6 pueden ser usadas como salidas PWM), 8 entradas analógicas y un cristal de 16 MHZ.



Figura 2. 8. Arduino Pro Mini

La Arduino Mini (ver Figura 2.8.) es una placa con un microcontrolador ATmega168 (datasheet). Tiene 14 E/S digitales (6 de las cuales se pueden utilizar como salidas PWM), 6 entradas analógicas, un resonador interno, botón de reseteo y agujeros para el montaje de tiras de pines. Se le puede montar una tira de 6 pines para la conexión a un cable FTDI o a una placa adaptadora de la casa Sparkfun para dotarla de comunicación USB y alimentación.



Figura 2. 9. USB/Serial Light Adapter

Es una placa básica que utiliza una interfaz RS232 para comunicarse con el ordenador o para la carga de sketches. Esta placa es fácil de montar, incluso como ejercicio de aprendizaje. Se ha diseñado para utilizar los componentes más simples posibles, de manera que sea fácil de construir, incluso si buscas las componentes en la tienda de la esquina.

Para obtener información de las placas más nuevas y actualizadas es necesario visitar la web en inglés[4], puesto que las hojas de características no están traducidas aún.

2.3. Arduino Mega 2560

2.3.1. Introducción

Arduino Mega 2560 (ver Figura 2.10.) es una placa electrónica basada en el microcontrolador *Atmega2560*.

Tiene 54 entradas/salidas digitales y 14 de éstas pueden utilizarse para salidas PWM. Además lleva 16 entradas analógicas, UARTs (puertos seriales), un oscilador de 16MHz, una conexión USB, un conector de alimentación, un header ICSP y un pulsador para el reset. La placa lleva todo lo necesario para soportar el microcontroladores.

Para empezar a utilizar la placa sólo es necesario conectarla al ordenador a través de un cable USB, o alimentarla con un adaptador de corriente AC/DC. También, para empezar, puede alimentarse sencillamente con una batería.

Una de las características principales de la MEGA 2560 es que no utiliza el convertidor USB-serial FTDI. Por el contrario, ofrece el microprocesador *Atmega8U2* programado como convertidor USB-serial.

La placa Arduino MEGA2560 es compatible con la mayoría de los shield sostenidos por las placas Duemilanove y UNO.



Figura 2. 10. Arduino Mega 2560

2.3.2. Características

- Microprocesador ATmega2560
- Tensión de alimentación (recomendado) : 7-12 V
- Integra regulación y estabilización de +5Vcc
- 54 líneas de Entradas/Salidas digitales (14 de ellas se pueden utilizar como salidas PWM)
- 16 entradas analógicas
- Máxima corriente continua para las entradas : 40 mA
- Salida de alimentación a 3.3V con 50 mA
- Memoria de programa de 256Kb (el bootloader ocupa 8Kb)
- Memoria SRAM de 8Kb para datos y variables del programa
- Memoria EEPROM para datos y variables no volátiles
- Velocidad del reloj del trabajo de 16MHz
- Reducidas dimensiones de 100x50 mm

2.3.3. Alimentación de la placa

Puede ser alimentado a través de la conexión USB o con una fuente de alimentación externa (ver Figura 2.11.). La fuente de alimentación se selecciona automáticamente.

Las fuentes de alimentación externas pueden ser o un transformador o una batería. El transformador se puede conectar usando un conector macho de 2.1mm con centro positivo en el conector hembra de la placa. Los cables de la batería pueden conectarse a los pines GND y VIN en los conectores de alimentación.

Fuente externa

USB BM

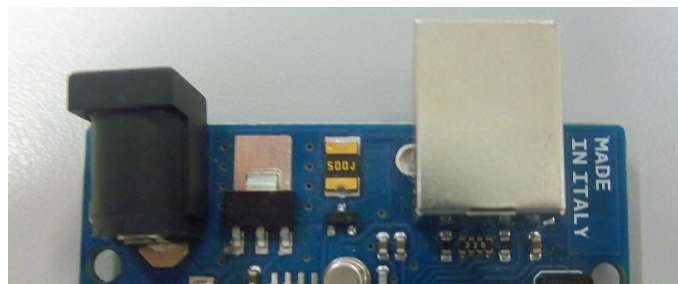


Figura 2. 11. Alimentación de Arduino

Se puede operar con un suministro externo de 6 a 20 voltios. Si se proporcionan menos de 7V, sin embargo, el pin de 5V puede proporcionar menos voltaje del deseado y la placa puede ser inestable. Si se utiliza más de 12V, el regulador de voltaje se puede sobrecalentar y dañar la placa. El rango recomendado es de 7 a 12 voltios.

Los pines de alimentación (ver Figura 2.12.) son:

VIN. Es la entrada de voltaje a la placa Arduino cuando se está usando una fuente externa de alimentación. Se puede proporcionar voltaje a través de este pin, o, si se está alimentado a través de la conexión de 2.1mm, acceder a ella a través de este pin.

5V. Es la fuente de voltaje estabilizado usado para alimentar el microcontrolador y otros componentes de la placa. Esta puede provenir de V_{IN} a través de un regulador integrado en la placa, o proporcionada directamente por el USB u otra fuente estabilizada de 5V. Hay un total de tres pines de 5V, uno en la zona de alimentación y otros dos en la zona de pines digitales.

3V3. Es una fuente de voltaje a 3,3 voltios generada en el chip FTDI integrado en la placa. La corriente máxima soportada es de 50mA.

GND. Pines de toma de tierra. Tiene un total de 5: dos en la zona de alimentación, dos en la zona de los pines digitales y otro más junto a los pines de PWM..



Figura 2. 12. Pines de alimentación

2.3.4. Pines de la placa

- **Pines digitales:** Cada uno de los 54 pines digitales en los Mega se puede utilizar como una entrada o salida, utilizando *pinMode()*, *digitalWrite()*, y las funciones *digitalRead()*. Funcionan a 5 voltios. Cada pin puede proporcionar o recibir un máximo de 40 mA y tiene una interna de pull-up resistor (desconectada por defecto) de 20-50 kOhmios.



Figura 2. 13. Pines digitales

- **Pines puerto serie:** Serie: 0 (RX) y 1 (TX); serie 1: 19 (RX) y 18 (TX); serie 2: 17 (RX) y 16 (TX); serie 3: 15 (RX) y 14 (TX) . Se utiliza para recibir (RX) y transmitir (TX) datos serie TTL. Los pines 0 y 1 están también conectados a los pines correspondientes del ATmega16U2 USB-to-Serial TTL chips.

-**Pines interrupciones:** Se pueden encontrar distribuidos por los pines de la placa Arduino. Las interrupciones externas se corresponden con los pines: 2 (interrupción 0), 3 (interrupción 1), 18 (interrupción 5), 19 (interrupción 4), 20 (interrupción 3), y 21 (interrupción 2). Estos pines se pueden configurar para lanzar una interrupción en un valor LOW (0V), en flancos de subida o bajada (cambio de LOW a HIGH (5V) o viceversa), o en cambios de valor.

- **Pines PWM:** 2 a 13 y 44 a 46. Proporcionar 8-bit de salida PWM con la función *analogWrite*

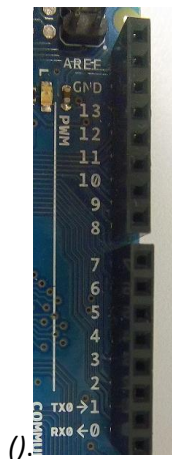


Figura 2. 14. Pines PWM

- **Pines analógicos:** El Mega2560 tiene 16 entradas analógicas, cada una de las cuales proporcionan 10 bits de resolución (es decir, 1024 valores diferentes). Por defecto se mide desde tierra a 5 voltios, aunque es posible cambiar el extremo superior de su rango usando el pin AREF y la función *analogReference* ().

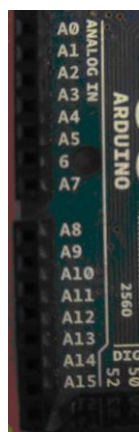


Figura 2. 15. Pines analógicos

- Otros pines de la placa:

AREF. Voltaje de referencia para las entradas analógicas. Se utiliza con *analogReference ()*.



Figura 2. 16. Pin AREF

Reset. Línea LOW para reiniciar el microcontrolador. Normalmente se utiliza para agregar un botón de reinicio.

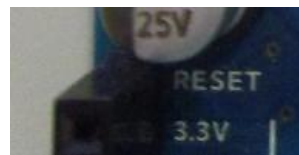


Figura 2. 17. Pin RESET

SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS). Estos pines apoyan la comunicación SPI con la biblioteca de SPI.

LED 13: Hay un LED integrado conectado al pin digital 13. Cuando el pin es de alto valor, el LED está encendido, cuando el pasador es bajo, está apagado.

2.3.5. Memoria

El *Atmega2560* tiene 256 KB de memoria flash para almacenar el código (de 8 KB que se utiliza para el cargador de arranque), 8 KB de SRAM y 4 KB de memoria EEPROM.

2.3.6. Comunicación

El Arduino Mega2560 tiene un número de instalaciones para la comunicación con un ordenador, otro Arduino, u otros microcontroladores. El *Atmega2560* proporciona cuatro UART hardware para TTL (5V) de comunicación serie. Proporcionan un puerto COM virtual con el software en el equipo (máquinas con Windows se necesita un archivo.inf, pero en máquinas OSX y Linux reconoce la tarjeta como un puerto COM automáticamente). El software de Arduino incluye un monitor de serie que permite simples datos de texto que se envían desde y hacia la placa. Los LEDs RX y TX de la placa parpadea cuando los datos se transmiten a través de la *ATmega8U2/ATmega16U2* conexión chip y USB al ordenador (pero no para la comunicación en serie en los pines 0 y 1).

Una biblioteca `SoftwareSerial` permite la comunicación serial en cualquiera de los pines digitales de la Mega2560.

El *Atmega2560* también es compatible con TWI y la comunicación SPI. El software de Arduino incluye una librería `Wire` para simplificar el uso del bus TWI.

2.3.7. Programación

Los Arduino Mega pueden ser programados con el software Arduino [1].

El *Atmega2560* en el Arduino Mega viene con un cargador de arranque que le permite cargar nuevo código a la misma sin el uso de un programador de hardware externo. También puede pasar por alto el gestor de arranque y programar el microcontrolador a través del ICSP (programación In-Circuit Serial) [5].

2.4. Entorno de desarrollo para Arduino

El entorno de desarrollo para Arduino se encuentra disponible de forma gratuita en la página web de Arduino [2]. Se puede elegir la versión dependiendo del sistema operativo utilizado. Las versiones se van actualizando cada poco tiempo, y si existe una actualización disponible salta un aviso cada vez que se abre el entorno.

El entorno (ver Figura 2.18.) está constituido por un editor de texto para escribir el código, un área de mensajes, una consola de texto, una barra de herramientas con botones para las funciones comunes, y una serie de menús. Permite la conexión con el hardware de Arduino para cargar los programas y comunicarse con ellos.

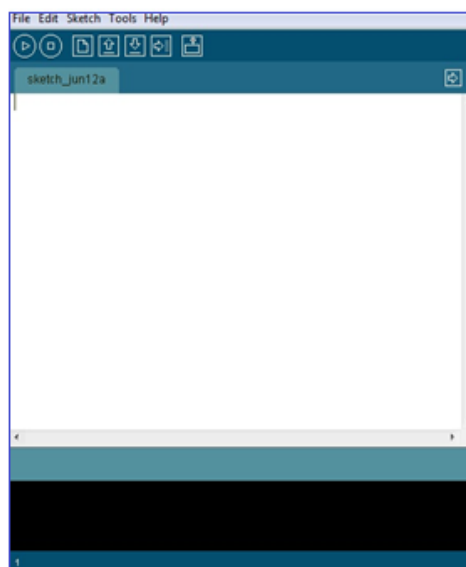


Figura 2. 18. Entorno Arduino

Para conectar la placa con el entorno de desarrollo, se necesita seleccionar el tipo de placa, como se ve en la Figura 2.19.

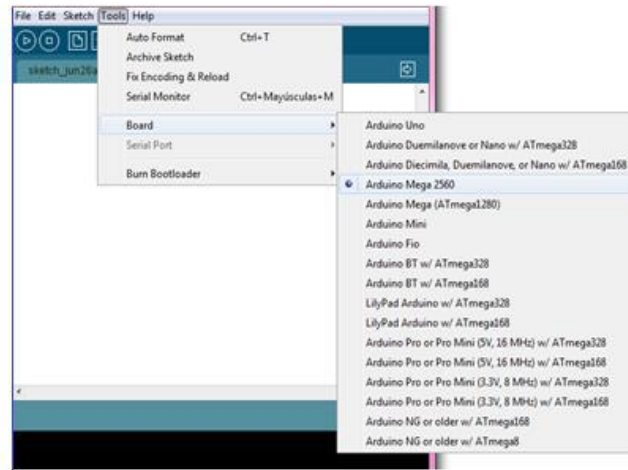


Figura 2. 19. Selección de la placa

Una vez seleccionada la placa es necesario seleccionar el puerto serie en el que se encuentra, para poder comenzar la comunicación. El número del puerto serie cambia si se encuentra en Linux o en Windows. En este caso en Windows se encuentra en el **COM3**, mientras que en Linux, pese a poder cambiar, suele encontrarse en el **/dev/ttyACM0**. En la Figura 2.20 y en la Figura 2.21. Se muestran la selección del puerto serie.

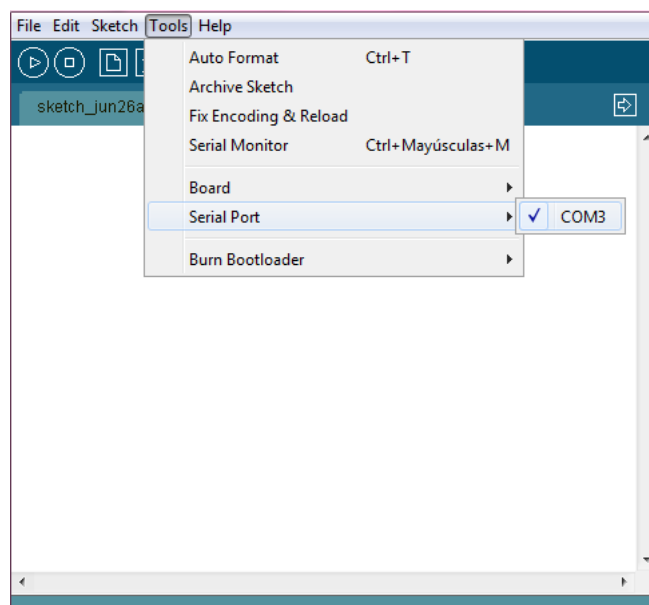


Figura 2. 20. Selección de puerto en Windows

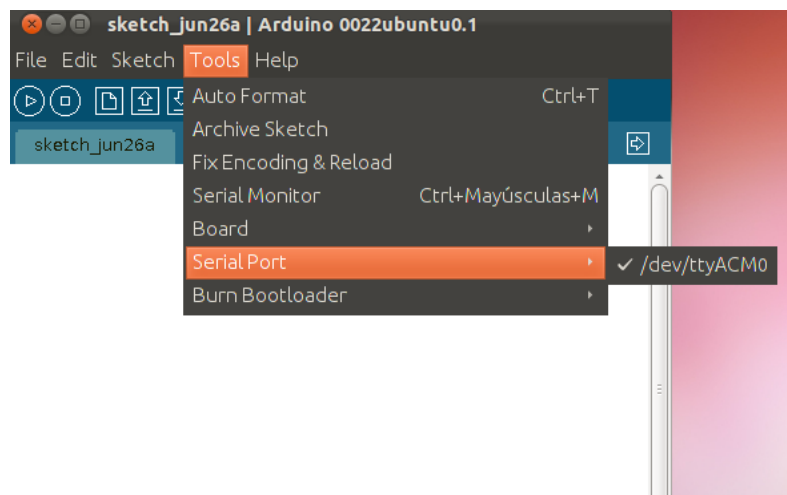


Figura 2. 21. Selección de puerto en Linux

Arduino utiliza para escribir el software lo que denomina "sketch" (programa). Estos programas son escritos en el editor de texto. En el área de mensajes se muestra información mientras se cargan los programas y también muestra los errores ocurridos al compilar o cargar el programa, o si estos procesos se han realizado satisfactoriamente. La consola muestra el texto de salida para el entorno de Arduino incluyendo los mensajes de error completos y otras informaciones. La barra de herramientas (ver Figura 2.22.) permite verificar el proceso de carga (compilar), pararlo, creación, apertura y guardado de programas, descargar el programa en la placa y la monitorización serie.



Figura 2. 22. Barra de herramientas

Para empezar a programar en Arduino, a parte de la información encontrada en su página web [2], se pueden consultar manuales escritos para tal efecto [6].

2.4.1. Librerías de Arduino

El entorno de programación de Arduino cuenta con una gran variedad de librerías. Para poder usarlas simplemente se deben añadir al principio del programa, tal como se muestra en la Figura 2.23.

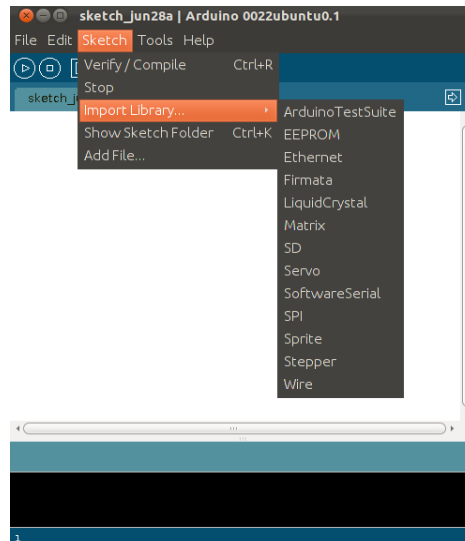


Figura 2. 23. Selección de librerías

En la web de Arduino existen más librerías que pueden ser descargadas gratuitamente [7]. Una vez que se han descargado, deben introducirse en el entorno de programación, en la carpeta destinada para tal efecto, además de introducirlas en el hardware de Arduino. Todo este proceso se encuentra explicado en el apartado de librerías de la página web de Arduino anteriormente mencionada.

Capítulo 3: QT

3.1. Introducción

Qt es un entorno multiplataforma ampliamente usado para desarrollar aplicaciones con interfaz gráfica de usuario, así como también para el desarrollo de programas sin interfaz gráfica, como herramientas para la línea de comandos y consolas para servidores.

Qt se desarrolla como un software libre y de código abierto a través de Qt Project [8] , donde participa tanto la comunidad, como desarrolladores de Nokia, Digia y otras empresas. Qt es utilizada en KDE, entorno de escritorio para sistemas como GNU/Linux o FreeBSD, entre otros.

Qt utiliza el lenguaje de programación C++ como principal, adicionalmente puede ser utilizado en varios otros lenguajes de programación a través de “bindings”. También se usa en sistemas informáticos empleados para automoción, aeronavegación y aparatos domésticos; como frigoríficos.

Funciona en todas las principales plataformas, y tiene un amplio apoyo. El API de la biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML, gestión de hilos, soporte de red, una API multiplataforma unificada para la manipulación de archivos y una multitud de otros para el manejo de ficheros, además de estructuras de datos tradicionales.

Tiene una página web [9] donde se puede encontrar toda la información sobre QT en inglés. Además se puede descargar gratuitamente desde esa misma página web [10] y existe un enlace [11] donde puedes enviar preguntas o sugerencias.

Algunos ejemplos de aplicaciones populares que utilizan Qt son; Adobe Photoshop Album, Google Earth, Mathematica, Qt Creator, Skype etc.

3.2. Plataformas

Qt se encuentra disponible para sistemas tipo unix con el servidor gráfico X Window System (Linux, BSDs, Unix), para Apple Mac OS X, para sistemas Microsoft Windows, para Linux embebido (en inglés Embedded Linux, para sistemas embebidos como PDA, Smartphone, etc.) y para dispositivos que utilizan Windows CE.

Adicionalmente también está disponible QSA (Qt Scripts for Applications), que, basándose en ECMAScript/JavaScript, permite introducir y crear scripts en las aplicaciones creadas con Qt.

Hay tres ediciones de Qt disponibles en cada una de estas plataformas, llamadas:

- GUI Framework – edición con nivel reducido de GUI, orientado a redes y bases de datos.
- Full Framework – edición completa comercial.
- Open Source – edición completa Open Source.

3.3. Bindings

Qt dispone de una serie de bindings para diversos lenguajes de programación:

- PyQt – Bindings GPL/Comercial para Python.
- PySide – LGPL bindings para Python de OpenBossa (subsidiario de Nokia).
- PythonQt – LGPL bindings para Python.
- Qyoto – Bindings para C# u otros lenguajes.NET. Existe un conjunto adicional de bindings Kimono para KDE.
- QtRuby – Bindings para Ruby. Existe un conjunto adicional de bindings, Korundum para KDE.
- Qt Jambi – Bindings para Java.
- QtAda – Bindings para Ada.
- FreePascal Qt4 – Bindings para Pascal.
- Perl Qt4– Bindings para Perl.
- PHP-Qt – Bindings para PHP.
- Qt Haskell– Bindings para Haskell.
- lqt – Bindings para Lua.
- DaoQt – Bindings para Dao.
- QtD – Binding para D.

3.4. QT Creator

3.4.1. Introducción

Qt Creator es un entorno de desarrollo integrado (IDE) (ver Figura 3.1.) que proporciona las herramientas necesarias para diseñar y desarrollar aplicaciones con el marco de aplicaciones Qt.

Qt está diseñado para desarrollar aplicaciones e interfaces de usuario una vez y desplegarlas a través de varios escritorios y sistemas operativos móviles. Qt Creator proporciona herramientas para llevar a cabo sus tareas a lo largo de todo el desarrollo de aplicaciones de ciclo de vida, desde la creación de un proyecto para desplegar la aplicación en las plataformas de destino.

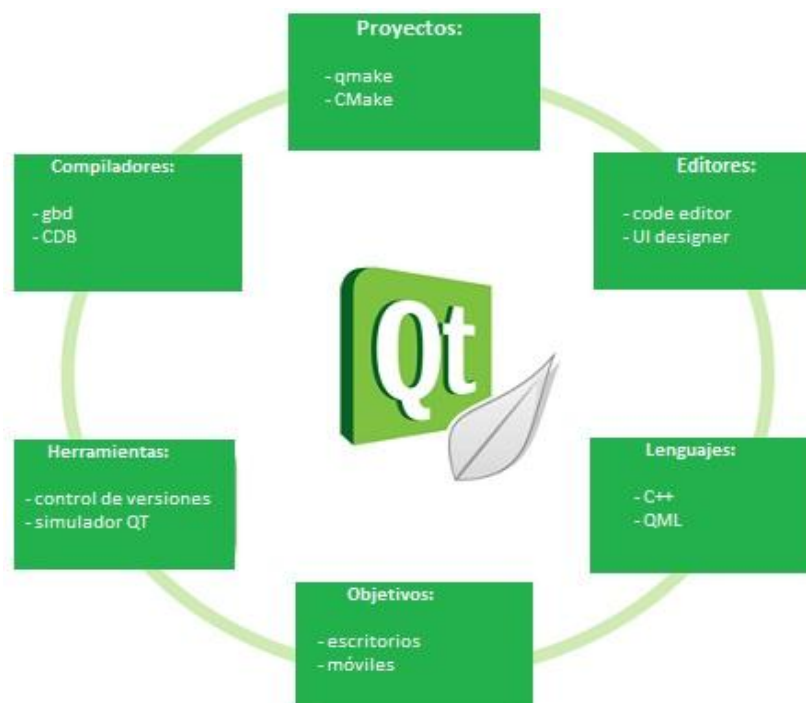


Figura 3. 1. Qt Creator[5]

Uno de las mayores ventajas de Qt Creator es que permite que un equipo de desarrolladores comparta un proyecto a través de diferentes plataformas de desarrollo (Microsoft Windows, Mac OS X, y Linux) con una herramienta común para desarrollo y depurado.

El objetivo principal de Qt Creator es satisfacer las necesidades de desarrollo de los desarrolladores de Qt que buscan simplicidad, facilidad de uso, productividad, extensibilidad y apertura. Las características clave de Qt Creator permiten a los desarrolladores realizar las siguientes tareas:

- Comenzar con el desarrollo de aplicaciones Qt rápida y sencilla con asistentes de proyectos, y acceder rápidamente a los últimos proyectos y sesiones.
- Diseño Qt widget de una aplicación basada en la interfaz de usuario con el editor integrado, Qt Designer.
- Desarrollar aplicaciones con el avanzado editor de código C++ que proporciona nuevas características de gran alcance para completar fragmentos de código, código de refactorización, y ver el contorno de los archivos (es decir, la jerarquía de símbolos de un archivo).
- Crear, ejecutar y desplegar proyectos de Qt que se dirigen a múltiples plataformas de escritorio y móviles, tales como Microsoft Windows, Mac OS X, Linux, Symbian, MeeGo y Maemo.
- Depurar con los depuradores GNU y el CDB mediante una interfaz gráfica de usuario con mayor conocimiento de la estructura de clases Qt.
- Utilizar las herramientas de análisis de código para comprobar si existen problemas de gestión de memoria en las aplicaciones.
- Implementar aplicaciones para dispositivos móviles y crear paquetes de instalación de aplicaciones para Symbian, MeeGo y Maemo dispositivos que se pueden publicar en la tienda Ovi y otros canales.
- Acceder fácilmente a la información con el módulo contextual sistema de ayuda Qt.

Qt Creator permite la colaboración entre los diseñadores y desarrolladores. Los diseñadores trabajan en un entorno visual, mientras que los desarrolladores trabajan en un IDE fullfeatured y Qt Creator soporta iteración de ida y vuelta desde el diseño hasta el código, probar y volver a diseñar.

3.4.2. Características

Las características clave de Qt Creator permiten a los programadores realizar las siguientes tareas:

- Empezar a desarrollar aplicaciones con Qt rápida y fácilmente con el asistente de proyectos, y acceder rápidamente a proyectos recientes y sesiones.
- Diseñar aplicaciones de interfaz de usuario basadas en widgets Qt con el editor integrado, Qt Designer.
- Desarrollar aplicaciones con el editor de código avanzado de C++ que provee nuevas y poderosas características para completar recortes de código, remanufactura de código, y viendo el esquema de archivos (que es, la jerarquía de símbolos de un archivo)
- Compilar, correr y desarrollar proyectos Qt que se dirigen a múltiples plataformas de escritorio y móviles, como Microsoft Windows, Mac OS X, Linux, Symbian, MeeGo, y Maemo.
- Depurar con el depurador GNU y el CDB usando una interfaz gráfica de usuario con una mayor conciencia de las estructuras de clases Qt.
- Usar herramientas de análisis de código para verificar la administración de memoria en sus aplicaciones.
- Desarrollar aplicaciones para dispositivos móviles y crear paquetes de instalación para dispositivos Symbian, MeeGo, y Maemo que pueden ser publicadas en la tienda Ovi y otros canales.
- Acceder fácilmente a información con el módulo del sistema de ayuda contextual de Qt.

3.4.3. Sistemas operativos soportados

El paquete de instalación de Qt Creator está disponible para Microsoft Windows, Mac OS X, y Linux. Qt Creator puede ser ejecutado en otras plataformas, pero requiere la compilación del código fuente disponible públicamente. Compilar y correr Qt Creator desde el código fuente puede requerir una instalación separada de Qt en tu computador.

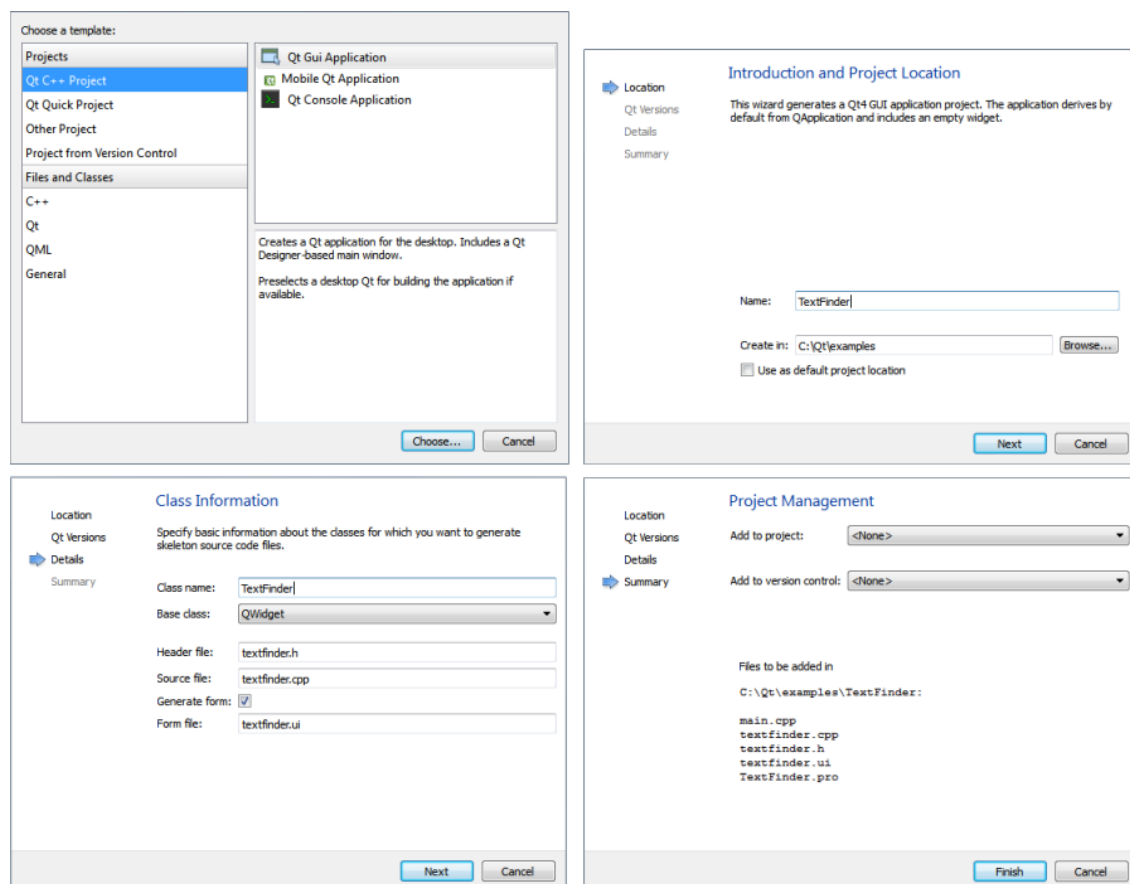
3.4.4. Gestión de proyectos

Una de las principales ventajas de Qt Creator es que permite a un equipo de desarrolladores compartir un proyecto a través de diferentes plataformas de desarrollo con una herramienta común para el desarrollo y la depuración.

- Creación de proyectos

Para poder crear y ejecutar aplicaciones, Qt Creator necesita la misma información que un compilador. Esta información se especifica en el proyecto de generar y ejecutar ajustes.

La creación de un nuevo proyecto en Qt Creator es ayudado por un asistente de los desarrolladores de guías paso a paso a través del proceso de creación del proyecto. En la primera etapa, los desarrolladores seleccionan el tipo de proyecto, desde las categorías: Qt C++, un proyecto Qt Quick, u otro proyecto. A continuación, los desarrolladores pueden seleccionar una ubicación para el proyecto y especificar la configuración de la misma.



The figure displays four sequential screenshots of the Qt Creator project creation wizard:

- Choose a template:** The 'Qt C++ Project' template is selected under the 'Projects' category. The description states: 'Creates a Qt application for the desktop. Includes a Qt Designer-based main window. Presselects a desktop Qt for building the application if available.'
- Introduction and Project Location:** The wizard generates a Qt4 GUI application project. The 'Name' field is 'TextFinder' and the 'Create in' location is 'C:\Qt\examples'. The 'Use as default project location' checkbox is unchecked.
- Class Information:** The user specifies basic information about the classes. The 'Class name' is 'TextFinder', the 'Base class' is 'QWidget', and the 'Generate form' checkbox is checked. The source files listed are 'textfinder.h', 'textfinder.cpp', and 'textfinder.ui'.
- Project Management:** The user can add the project to an existing project or version control. The 'Files to be added in' section lists the files: 'main.cpp', 'textfinder.cpp', 'textfinder.h', 'textfinder.ui', and 'TextFinder.pro'.

Figura 3. 2. Nueva Qt GUI proyecto aplicación asistente

Cuando los pasos se han completado, Qt Creator genera automáticamente el proyecto con encabezados requeridos, archivos fuente, las descripciones de la interfaz de usuario y los archivos del proyecto, tal como se define por el asistente.

No sólo el asistente ayuda a los nuevos usuarios a empezar a trabajar rápidamente, sino que también permite a los usuarios más experimentados optimizar su flujo de trabajo para la creación de nuevos proyectos. La interfaz de usuario conveniente hace más fácil asegurar que un proyecto comienza con la configuración correcta y dependencias.

- Control de versiones

La forma recomendada de configurar un proyecto es utilizar un sistema de control de versiones. Sólo los archivos de código fuente del proyecto deben ser almacenados y los archivos generados por el sistema de construcción o Qt Creator no deben ser almacenados.

Qt Creator soporta un número de sistemas de control de versiones, integrando su utilización en el entorno de trabajo. Los sistemas compatibles incluyen Bazaar, CVS, Git, Mercurial, Perforce y Subversion.

La configuración es sencilla, con valores comunes para el control de versiones y características específicas de los sistemas de control de versiones localizadas herramientas sub-menús.

La salida para cada sistema se muestra en el panel de control de versión de salida. Los elementos de interfaz de usuario para la visualización y gestión de repositorios están disponibles para algunos sistemas.

- Configurar los proyectos

Qt Creator permite especificar distintas configuraciones de compilación para cada plataforma de desarrollo. De forma predeterminada, las versiones de sombra se utilizan para mantener los archivos de creación específicos separados de la fuente. Se pueden crear versiones independientes de los archivos de proyecto para mantener dependiente de la plataforma de código independiente.

- Administración de sesiones

Los artículos tales como los archivos abiertos, puntos de interrupción, y las expresiones evaluadas se almacenan en las sesiones. No se consideran parte de la información compartida entre plataformas.

3.4.5. Diseño de interfaces de usuario

Qt Creator proporciona dos editores visuales integrados, Qt Quick y Qt Designer. La integración incluye la gestión de proyectos y finalización del código.

- Desarrollo de aplicaciones Qt Quick

Crear proyectos de Qt Quick desde cero o importar los proyectos existentes para Qt Creator. Se puede utilizar el editor de código (modo de edición) o el editor visual (modo de diseño) para desarrollar aplicaciones Qt Quick.

- Desarrollo de aplicaciones basadas en Widget

Widgets y formas creadas con Qt Designer se integran a la perfección con código programado mediante el uso de las señales de Qt y el mecanismo de franjas horarias que le permite asignar fácilmente el comportamiento de los elementos gráficos. Todas las propiedades establecidas en Qt Designer se pueden cambiar de forma dinámica dentro del código. Por otra parte, las características tales como la promoción widget y plugins personalizados permiten utilizar sus propios widgets con Qt Designer.

- Optimizar aplicaciones para dispositivos móviles

Los dispositivos móviles se han diseñado para su uso. Hay que mantener las características de los dispositivos móviles a la hora de crear aplicaciones para ellos.

3.4.6. Codificación

Redacción, edición y navegación en el código fuente, son tareas fundamentales en el desarrollo de aplicaciones. Por lo tanto, el editor de código es uno de los componentes clave de Qt Creator. El editor de código se puede utilizar en el modo de edición para escribir código.

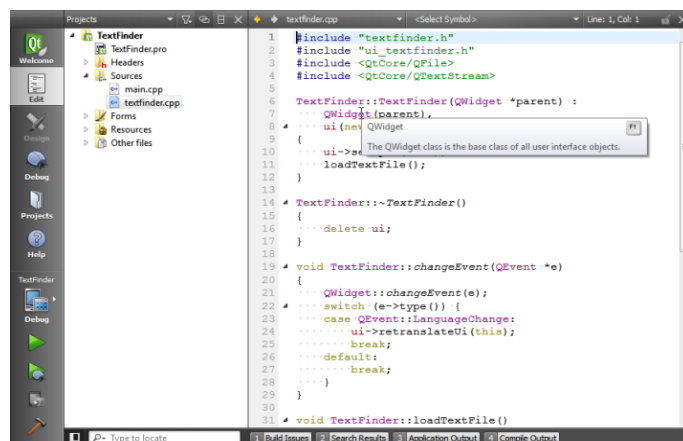


Figura 3. 3. Editor de código en modo de edición

El editor de código ofrece una serie de características que ayudan a los desarrolladores a mantener la legibilidad y el estilo de codificación:

- El resultado de sintaxis para las palabras clave, símbolos y archivos en C + +.
- Código de finalización de elementos, propiedades, identificadores y fragmentos de código.
- Verifica la sintaxis de código y se marcan los errores durante la edición, por lo que no es necesario utilizar la compilación simplemente como una manera de encontrar errores de ortografía y errores de sintaxis.
- Auto-sangría para el diseño de código fuente.
- La capacidad para contraer y expandir funciones en el código fuente (el plegado de código).
- El Localizador de herramienta de navegación para un acceso rápido a los archivos, los símbolos, la jerarquía, y otra información.
- El apoyo a la refactorización del código para mejorar la calidad interna o en su aplicación, su rendimiento y extensibilidad, y la legibilidad del código y de mantenimiento, así como para simplificar la estructura del código.

Además de estas características, el editor de código tiene otras características útiles, tales como:

- Búsqueda incremental donde se destacan las cadenas coincidentes en la ventana mientras se escribe. Búsqueda avanzada que permite realizar búsquedas de los proyectos abiertos o archivos en el sistema. -Además, puede buscar símbolos cuando se desea restaurar código.
- La numeración de líneas actuales y resultados de línea.
- Comentar fácilmente el código.
- El cambio rápido entre la definición y método de declaración de la función.
- Marcadores para facilitar la navegación en el código.

El editor de código soporta atajos de teclado para diferentes opciones de edición más rápida. Es posible trabajar sin necesidad de utilizar el ratón en absoluto, lo que permite a los desarrolladores mantener sus manos en el teclado y trabajar de manera más productiva.

3.4.7. Creación y ejecución

Qt Creator proporciona apoyo para la creación y ejecución de aplicaciones Qt para entornos de escritorio (Windows, Linux, y Mac OS) y dispositivos móviles (Symbian, MeeGo, Maemo).



Figura 3. 4. El mismo código fuente construido se ejecuta en múltiples objetivos

Qt Creator permite a los desarrolladores especificar diferentes configuraciones de compilación para cada plataforma de desarrollo y para cambiar rápidamente a la hora de construir objetivos. De forma predeterminada, las versiones instantáneas se utilizan para mantener los archivos de creación específicos separados de la fuente. Los desarrolladores pueden crear versiones independientes de los archivos de proyecto para mantener dependiente de la plataforma de código independiente. Pueden utilizar ámbitos qmake para seleccionar el archivo a procesar en función de la plataforma qmake se ejecuta.

Además de prestar apoyo a qmake, propia herramienta de compilación de Qt, Qt Creator también viene con soporte para CMake [cmake.org], una alternativa popular. CMake es una configuración multiplataforma y herramienta de construcción que trabaja con las cadenas de herramientas del compilador nativo de Microsoft Windows, Mac OS X, Linux y otras plataformas que soporta la herramienta. Sin embargo, sólo se admite el sistema de construcción para aplicaciones móviles en Qt Creator es qmake.

Qt Creator también apoya proyectos genéricos, donde los desarrolladores o bien utilizan un sistema de construcción sin apoyo, o no desea asociar un sistema de construcción con su proyecto en absoluto. En casos como estos, Qt Creator funciona como un editor de código y la configuración de generación puede ser especificada manualmente al proyecto.

3.5. Obtención de ayuda

De vez en cuando, los desarrolladores pueden necesitar información adicional acerca de una clase determinada, la función, o cualquier otra parte de la API de Qt. Toda la documentación Qt y los ejemplos son accesibles a través de plug-in de ayuda en Qt Creator.

Para ver la documentación, el modo de ayuda se utiliza, donde la mayor parte de la ventana se dedica al texto de ayuda. Al trabajar con el código fuente en el modo de edición, los desarrolladores pueden acceder a la ayuda sensible al contexto, moviendo el cursor de texto a una clase Qt o función y pulse la tecla F1. La documentación se muestra en un panel en el lado derecho del editor de código, como se muestra en la siguiente figura.

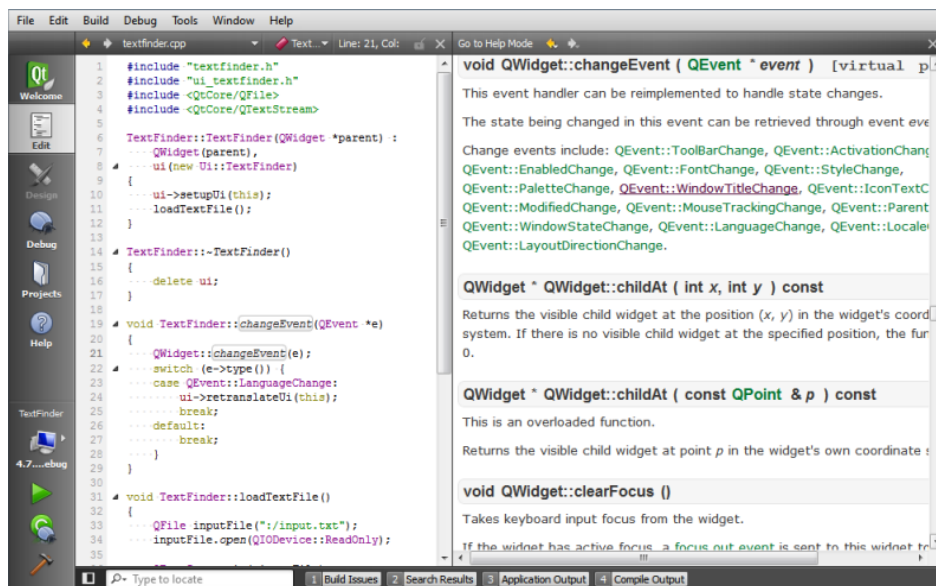


Figura 3. 5 Contexto sensible de ayuda Qt

También es posible añadir documentación externa a Qt Creator, complementar o sustituir la documentación existente según se requiera.

3.6. Usando el compilador Mega-Object (MOC)

3.6.1. Introducción

El compilador Mega-Object (MOC) es el programa que se encarga de Qt en extensiones C++ .

La herramienta MOC lee un archivo de cabecera C + +. Si encuentra una o más declaraciones de clases que contienen la **Q_OBJECT** , produce un archivo C + + de código fuente que contiene el código meta-objeto para esas clases. Entre otras cosas, meta-código objeto se requiere para las señales y el mecanismo de ranuras, la información de tipo de tiempo de ejecución, y el sistema de propiedad dinámica.

El archivo de C + + código fuente generado por MOC debe ser compilado y vinculado con la implementación de la clase.

3.6.2. Uso

MOC se suele utilizar con un archivo de entrada que contiene declaraciones de clase como esta:

```
class mi clase : public QPushButton
{
    Q_OBJECT
public:
    mi clase(QWidget *parent = 0);
signals:
    void misigna();
private slots:
    void mislot();
};
```

Además de las señales y ranuras mostradas anteriormente, MOC también implementa propiedades de objeto. La función *Q_PROPERTY()* declara una propiedad del objeto, mientras que *Q_ENUMS()* declara una lista de los tipos de enumeración dentro de la clase para ser utilizable en el interior del sistema de la propiedad .

La función *Q_FLAGS()* declara enumeraciones que se van a utilizar como banderas, es decir, operación lógica OR juntos. Otra función, *Q_CLASSINFO()*, permite conectar otros pares nombre/valor a la clase de meta-objeto.

La salida producida por MOC debe ser compilada y vinculada, al igual que todo código C + + en el programa, de lo contrario, la construcción fallará en la fase de enlace final. Si usa qmake, esto lo hace de forma automática. Cada vez que se ejecute qmake, analiza los archivos del proyecto de encabezado y genera las reglas para invocar MOC para los archivos que contienen una función *Q_OBJECT*.

Si la declaración de clase se encuentra en el archivo *MyClass.h*, la salida MOC debe ser puesta en un archivo llamado *moc_myclass.cpp*. Este archivo debería entonces ser compilado, como de costumbre, por ejemplo, *moc_myclass.obj* en Windows. Este objeto se debe incluir en la lista de archivos de objeto que están unidos entre sí en la fase de construcción final del programa.

3.6.3. Escribir un reglamento para invocar MOC

Para cualquier cambio en los programas más sencillos de prueba, se recomienda que se automatice la ejecución del MOC. Mediante la adición de algunas reglas para makefile del programa, puede ejecutarse MOC cuando sea necesario

Se recomienda utilizar la herramienta de generación Trolltech makefile (qmake) para la construcción de makefiles. Esta herramienta genera un makefile que hace todo el manejo.

3.6.4. Diagnóstico

MOC advierte acerca de una serie de construcciones ilegales o peligrosas en las declaraciones de clase *Q_OBJECT*.

Si se obtienen errores de ligamiento en la fase de construcción final del programa, diciendo que *YourClass :: className ()* es indefinido o que *YourClass* carece de un mal vtable, suele significar que nos hemos olvidado de compilar o incluir el # moc-generado código C++, o (en el primer caso) que son objeto de archivo en el comando link. Si se usa qmake, hay que intentar volver a ejecutar para actualizar el archivo MAKE.

3.6.5. Limitaciones

MOC no maneja todo de C++. El principal problema es que las plantillas de clase no pueden tener señales o ranuras.



Capítulo 4: ROS

4.1. Introducción

ROS (Robot Sistema operativo) es un código abierto, meta sistema operativo para robots. Proporciona los servicios que se esperan de un sistema operativo, como abstracción de hardware de bajo nivel de control del dispositivo, la implementación de la funcionalidad de uso común, paso de mensajes entre procesos y gestión de paquetes. También proporciona herramientas y bibliotecas para obtener, construir, escribir y ejecutar código en varios equipos. ROS es similar en algunos aspectos a los marcos de robots, como jugador, Yarp , Orocos , CARMEN, Orca , MOOS, y Microsoft Robotics Studio.

El tiempo de ejecución “grafico” de ROS es una red peer-to-peer de procesos que están débilmente acoplados utilizando la infraestructura de comunicación ROS.

ROS implementa varios estilos diferentes de comunicación, incluyendo sincrónico estilo RPC de comunicación a través de los servicios , la transmisión de datos a través de asíncronas de los temas , y el almacenamiento de datos en un servidor de parámetros .

ROS no es un marco de tiempo real, aunque es posible la integración de ROS con el código de tiempo real.

ROS dispone de una página web[13] donde te puedes descargar gratuitamente el software[14] e instalarlo en tu ordenador. ROS ofrece a las bibliotecas y herramientas para ayudar a los desarrolladores de software a crear aplicaciones robóticas. Proporciona abstracción de hardware, los controladores de dispositivo, bibliotecas, visualizadores, paso de mensajes, gestión de paquetes, y mucho más. ROS está bajo licencia de código abierto (la licencia BSD).

ROS es una estructura distribuida de procesos (también conocidos como nodos) que permite a los ejecutables ser diseñada de forma individual y acoplamiento flexible en tiempo de ejecución. Estos procesos se pueden agrupar en paquetes y pilas, que pueden ser fácilmente compartidos y distribuidos. ROS también es compatible con un sistema federado de repositorios de código que permiten la colaboración a distribuir también. Este diseño, desde el nivel de sistema de archivos a nivel de la comunidad, permite tomar decisiones independientes sobre el desarrollo y la ejecución, pero todos se pueden reunir con herramientas de infraestructura ROS.

4.2. Objetivos del diseño

El objetivo principal de ROS es apoyar la reutilización de código en la investigación y desarrollo de robots. Su filosofía de trabajo podemos resumirla en los siguientes cinco puntos:

- **Peer-to-Peer**

Un sistema construido con ROS consiste en una serie de procesos que pueden encontrarse en máquinas diferentes, y que permanecen conectados en tiempo de ejecución mediante una topología Peer-to-Peer. Si los equipos de trabajo están conectados a una red heterogénea, no es necesario un servidor central para conseguir que todas las máquinas en servicio desarrollen los cálculos que requieren cada una de las tareas.

- **Multi-lenguaje**

ROS es capaz de trabajar en diferentes lenguajes de programación: C++, Python, Octave y LISP. En lugar de proporcionar una aplicación basada en C con interfaces de código auxiliar para la mayoría de los lenguajes, implementa éstas de forma nativa en cada lenguaje para seguir mejor las convenciones de cada uno de ellos.

- **Herramientas**

Con el objetivo de poder gestionar la complejidad de ROS, se ha optado por un diseño “microkernel”, donde numerosas herramientas se utilizan para generar y ejecutar diferentes operaciones como pueden ser:

- Navegar por el código fuente.
- Obtener y establecer los parámetros de configuración.
- Medir la utilización del ancho de banda.
- Graficar datos de mensajes.

Las ventajas que se consiguen son una mayor estabilidad y una reducción de la complejidad, por contra se pierde eficiencia.

- **Modular**

Debido a la dificultad que tiene la reutilización de código fuera del contexto original para el que fue creado, ROS dispone de librerías que permiten extraer código y reutilizarlo más allá de su intención original, siendo únicamente necesario crear pequeños ejecutables que facilitan la etapa de pruebas. Simuladores del proyecto Player, algoritmos de visión de OpenCV, algoritmos de planificación de OpenRAVE, son ejemplos entre muchos otros de esta reutilización de código abierto en forma de librerías.

- **Código libre y abierto**

El código fuente completo de ROS está disponible al público. Además, se distribuye bajo licencia BSD que permite tanto el desarrollo de proyectos comercializables como no comercializables. ROS pasa los datos entre módulos usando IPC (comunicación entre procesos) y no necesita que dichos módulos estén unidos al mismo ejecutable

4.3. Sistemas operativos

ROS en la actualidad sólo se ejecuta en plataformas basadas en UNIX. El software de ROS ha sido probado esencialmente en Ubuntu y Mac OS X, aunque la comunidad ROS ha contribuido el apoyo a Fedora, Gentoo, Arch Linux y otras plataformas Linux. Se está intentando incluir ROS en la plataforma WINDOWS.

4.4. Conceptos

ROS tiene tres niveles de conceptos: el nivel del sistema de archivos, el nivel de Computación Gráfica, y el nivel comunitario. Estos niveles y conceptos se resumen a continuación.

Además de los tres niveles de conceptos, ROS también define dos tipos de nombres: nombres de los paquetes de recursos y nombres Gráfico de recursos.

4.4.1. ROS nivel del sistema de archivos

Los conceptos de nivel de sistema de archivos son recursos que se encuentran en el disco, como por ejemplo:

- **Paquetes** : Los paquetes son la unidad principal para organizar el software en ROS. Un paquete puede contener procesos de ejecución ROS (nodos), una biblioteca ROS-dependiente, conjuntos de datos, archivos de configuración, o cualquier otra cosa que sea útil.
- **Manifiestos** : Manifiestos (*manifest.xml*) proporcionan metadatos sobre un paquete, incluyendo su información de licencia y dependencias, así como información específica del idioma, tales como banderas del compilador.
- **Pilas** : Pilas son colecciones de paquetes que proporcionan funcionalidad agregada, como una "pila de navegación."
- **Manifiestos de pila** : Los manifiestos de pila (*stack.xml*) proporcionan datos sobre una pila, incluyendo su información de licencia y sus dependencias en otras pilas.
- **Mensajes (msg); los tipos de mensajes**: descripciones, almacenados en *my_package / msg / MyMessageType.msg*, definen las estructuras de datos para los mensajes enviados en ROS.
- **Servicio (SRV) tipos de servicios**: descripciones, almacenadas en *my_package / srv / MyServiceType.srv*, definen la solicitud y estructuras de datos de respuesta de los servicios de ROS.

4.4.2. ROS a nivel de ejecución

ROS es una red peer-to-peer de procesos que están procesando los datos juntos. Los conceptos básicos de este nivel son nodos, maestro, servidor de parámetros, mensajes, servicios, temas y bolsas, los cuales proporcionan los datos de ejecución de diferentes maneras.

- **Nodos** : Los nodos son procesos que llevan a cabo cálculos. ROS está diseñado para ser modular en una escala de grano fino, un sistema de control de robot comprenderá usualmente muchos nodos. Por ejemplo, un nodo controla un telémetro láser, un nodo controla los motores de las ruedas, un nodo realiza localización, un nodo realiza la planificación de ruta, un nodo proporciona una vista gráfica del sistema, y así sucesivamente. Un nodo de ROS está escrito con el uso de bibliotecas de cliente ROS, tales como **roscpp** o **rospy**.
- **Maestro** : El Maestro ROS proporciona registro de nombres y la búsqueda para el resto de la ejecución de ROS. Sin el Maestro, los nodos no serían capaces de encontrar mensajes entre sí, intercambio, o invocar los servicios.
- **Parámetro Servidor** : El servidor de parámetros permite que los datos se almacenen con clave en un lugar central. En la actualidad es parte del Maestro.
- **Mensajes** : Los nodos se comunican entre sí pasando mensajes. Un mensaje es simplemente una estructura de datos, que comprende los campos con tipo. Los tipos estándar (entero, flotante, booleano, etc.) son compatibles. Los mensajes pueden incluir estructuras arbitrariamente anidadas y matrices (al igual que las estructuras de C).
- **Temas** : Los mensajes se enrutan a través de un sistema de transporte de publicación/suscripción semántica. Un nodo envía un mensaje mediante su publicación a un determinado tema. El tema es un nombre que se utiliza para identificar el contenido del mensaje. Un nodo que está interesado en un determinado tipo de datos se suscribirá al tema correspondiente. Puede haber varios editores y suscriptores concurrentes a un mismo tema, y un único nodo puede publicar y/o suscribirse a múltiples temas. En general, los editores y suscriptores no son conscientes de la existencia de los demás. La idea es disociar la producción de información a partir de su consumo. Lógicamente, se puede pensar en un tema como un bus de tipos de mensajes inflexibles. Cada bus tiene un nombre, y cualquier persona puede conectarse al bus para enviar o recibir mensajes, siempre y cuando sean del tipo correcto.
- **Servicios** : La publicación/suscripción es un modelo de comunicación no siempre flexible, es un medio de transporte que no es apropiado para las interacciones de petición/respuesta, que a menudo se requieren en un sistema distribuido. Para estos casos de petición/respuesta la comunicación se realiza a través de servicios, que se definen por un par de estructuras de mensajes: uno para la solicitud y uno para la respuesta. Un nodo proporciona un servicio bajo un nombre y un cliente utiliza el servicio enviando el mensaje de solicitud y esperando la respuesta. Las bibliotecas cliente de ROS presentan generalmente esta interacción para el programador como si fuera una llamada de procedimiento remoto.

- **Bolsas** : Las bolsas son un formato para guardar y reproducir datos de mensajes de ROS. Las bolsas son un mecanismo importante para el almacenamiento de datos, tales como datos del sensor, que puede ser difícil de recoger, pero es necesaria para desarrollar y probar algoritmos.

El Maestro ROS actúa como un servicio de nombres en la ejecución global de ROS. Almacena los temas y los servicios de información de registro de nodos. Los nodos pueden comunicarse con el maestro y reportar su información de registro. A medida que estos nodos se comunican con el Maestro, pueden recibir información sobre otros nodos registrados y realizar las conexiones, según corresponda. El Maestro también hará devoluciones de llamada a estos nodos cuando esta información cambia de registro, lo que permite a los nodos crear dinámicamente las conexiones cuando nuevos nodos se ejecutan.

Los nodos pueden conectarse a otros nodos directamente, el Maestro sólo proporciona información de búsqueda, al igual que un servidor DNS. Los nodos que se suscriben a un tema solicitarán conexiones de los nodos que publican ese tema, y establecen esa conexión a través de un acuerdo sobre el protocolo de conexión. El protocolo más común usado en un ROS se llama TCPROS, que utiliza el estándar TCP/IP sockets.

Esta arquitectura permite un funcionamiento desacoplado, donde los nombres son los principales medios por los cuales los sistemas más grandes y complejos pueden ser construidos. Los nombres tienen un papel muy importante en ROS: nodos, temas, servicios, y todos los parámetros tienen nombre. Cada biblioteca de cliente ROS soporta una línea de comandos de reasignación de nombres, lo que significa que un programa compilado puede ser reconfigurado en tiempo de ejecución para operar en una topología de computación gráfica diferente..

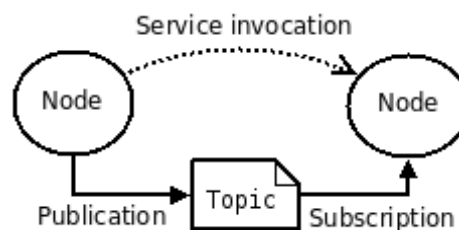


Figura 4. 1 Computación a nivel gráfico ROS

4.4.3. ROS a nivel comunitario

Los conceptos ROS a nivel comunitario son recursos que permiten a las comunidades separadas al software de intercambio y conocimiento. Estos recursos incluyen:

- **Distribución** : reparto de ROS, son colecciones de pilas versionadas que se pueden instalar. Distribuciones que juegan un papel similar al de las distribuciones de Linux: hacen que sea más fácil para instalar un conjunto de programas informáticos, y también mantener versiones consistentes a través de un conjunto de software.
- **Repositorios** : ROS se basa en una red federada de repositorios de código, donde diferentes instituciones pueden desarrollar y lanzar sus propios componentes de software del robot.

- **El Wiki ROS** : La comunidad Wiki de ROS [13] es el foro principal para documentar la información. Cualquier persona puede inscribirse para una cuenta y contribuir con su propia documentación, facilitar las correcciones o actualizaciones, escribir tutoriales y más.
- **Sistema Bug entradas.**
- **Listas de correo[17]:** La lista de correo ros-users es el principal canal de comunicación acerca de las nuevas actualizaciones de ROS, así como un foro para hacer preguntas sobre el software ROS.
- **ROS Respuestas[18]:** un sitio de preguntas y respuestas para contestar a sus preguntas relacionadas con ROS.

4.5. Software. Bibliotecas de clientes

Una biblioteca de cliente de ROS es una colección de código que facilita la tarea del programador ROS. Toma muchos de los conceptos de ROS y los hace accesibles a través de código. En general, estas bibliotecas permiten escribir nodos ROS, publicar y suscribirse a los temas , escribir y llamar a los servicios , y usar el parámetro del servidor. Tal biblioteca se puede implementar en cualquier lenguaje de programación, aunque el enfoque actual es proporcionar apoyo en C ++ y Python.

4.5.1. Bibliotecas principales de clientes

Toda la información referente a las bibliotecas centrales de clientes pueden leerse en la página web [15].

- **roscpp** : es una biblioteca de C ++ para el cliente ROS. Es la biblioteca de cliente ROS más utilizada y está diseñada para ser la biblioteca de más rendimiento para ROS.
- **rospy** : es una biblioteca de cliente para Python. Está diseñada para ofrecer las ventajas de un lenguaje orientado a objetos de secuencias de comandos de ROS. El diseño de rospy favorece la velocidad de ejecución (es decir, tiempo desarrollador) sobre el rendimiento en tiempo de ejecución para que los algoritmos puedan ser un prototipo y se pruebe rápidamente dentro de ROS. Muchas de las herramientas de ROS están escritas en rospy para tomar ventaja de las capacidades de tipo introspección.
- **roslisp** : es una biblioteca de cliente para LISP y se está utilizando actualmente para el desarrollo de bibliotecas de planificación. Es compatible tanto con la creación de nodos independientes como para el uso interactivo en marcha de un sistema de ROS.

4.5.2. Bibliotecas experimentales de clientes

- **rosjava** : es una implementación de ROS en lenguaje Java con soporte Android.
- **roslua** : es una biblioteca cliente para Lua, que es un ligero pero potente lenguaje. Esta biblioteca de cliente se encuentra actualmente en una etapa de desarrollo experimental y activo.
- **IOCN** : es una biblioteca cliente para Mono/.NET. Puede ser utilizada por cualquier lenguaje Mono/.NET, incluyendo C ++, Python Hierro, Hierro Ruby. El sistema de construcción de ROS creara archivos .DLL y .ETC para cada paquete escrito en los IOCN.

4.6. Ventajas e inconvenientes de ROS

ROS destaca por su código abierto y licencia libre BSD, que cuenta con una amplia base de documentación, tutoriales y soporte técnico. La fuerza de ROS se ve reflejada en la capacidad de hacer frente a diferentes aplicaciones del ámbito de la robótica.

- Visualización.
- Reconocimiento de objetos.
- Navegación.
- Manipulación.

VENTAJAS:

- Reduce el tiempo invertido en infraestructura y se centra en la investigación.
- Aborda problemas de alto nivel.
- Acelera el aprendizaje.
 - Viendo código de otros.
 - Viendo documentación de otros.
- Fomenta el trabajo en equipo y establece convenios, procesos y metodologías para hacer
- software reusable.

DESVENTAJAS:

- Sus objetivos son muy ambiciosos.
- Tratan con software muy variado (librerías hechas según criterio del autor).
- La integración de las aplicaciones no es inmediata.
 - Hay que leer suficiente documentación para comprender el funcionamiento.
 - Se debe revisar el código hecho, pues cada aplicación es distinta.
- Es necesaria una etapa de depuración.
- Exponer una librería en ROS no es gratis.
- Continuos cambios y evolución, que pueden dar como resultado un software obsoleto.
- Por ahora no tiene soporte para Windows u otros sistemas empujados

4.7. Conclusión

ROS es un experimento de ingeniería de software que permite la facilidad de desarrollo, pudiendo los investigadores centrarse en su área específica.

Su diseño abierto permite que pueda ser ampliado y aprovechado para construir diferentes sistemas softwares para robots, útiles para una gran variedad de plataformas hardware investigación y requerimientos de tiempo de ejecución.

Capítulo 5: IMPLEMENTACIÓN

Arduino-ROS

5.1. Introducción

En este capítulo se aborda la integración del software de ROS mediante su librería `rosserial` con Arduino. Una vez instalada y agregada al directorio de librerías Arduino (mediante el cuaderno de notas) aparecerá como librería **`ros_lib`** en el IDE Arduino. Esta librería permite la comunicación con ROS.

5.2. Librería Rosserial

El IDE Arduino y Arduino son herramientas para el hardware de programación rápida y sencilla. Usando el paquete **`rosserial_arduino`** [16], puede utilizar ROS directamente con el IDE Arduino.

Rosserial ofrece un protocolo de comunicación que funciona sobre ROS UART su Arduino. Permite que Arduino a ser un nodo de pleno derecho ROS que directamente pueda publicar y suscribirse a los mensajes de ROS, publicar y transformar TF, y obtener la hora del sistema ROS.

Nuestros enlaces ROS se implementan como una biblioteca Arduino. Al igual que todas las bibliotecas de Arduino, **`ros_lib`** trabaja poniendo su implementación de la biblioteca en la carpeta de *bibliotecas*.

Para utilizar las bibliotecas `rosserial` en el propio código, primero se debe poner

```
#Include<ros.h>
```

Antes de incluir los archivos de cabecera, por ejemplo, otros

```
#Include<std_msgs/String.h>
```

De lo contrario el IDE Arduino no será capaz de localizarlos.

Cuando se ha instalado ya sea de la fuente o través del comando `sudo`, lo único que hay que hacer es copiar el `roscd roserial_arduino/libraries` cp-r `ros_lib<sketchbook>/libraries/ros_lib`

roscd roserial_arduino/libraries cp-r ros_lib<sketchbook>/libraries/ros_lib

Después de reiniciar el IDE, debería aparecer *ros_lib* en ejemplos tal como se muestra en la figura 5.1

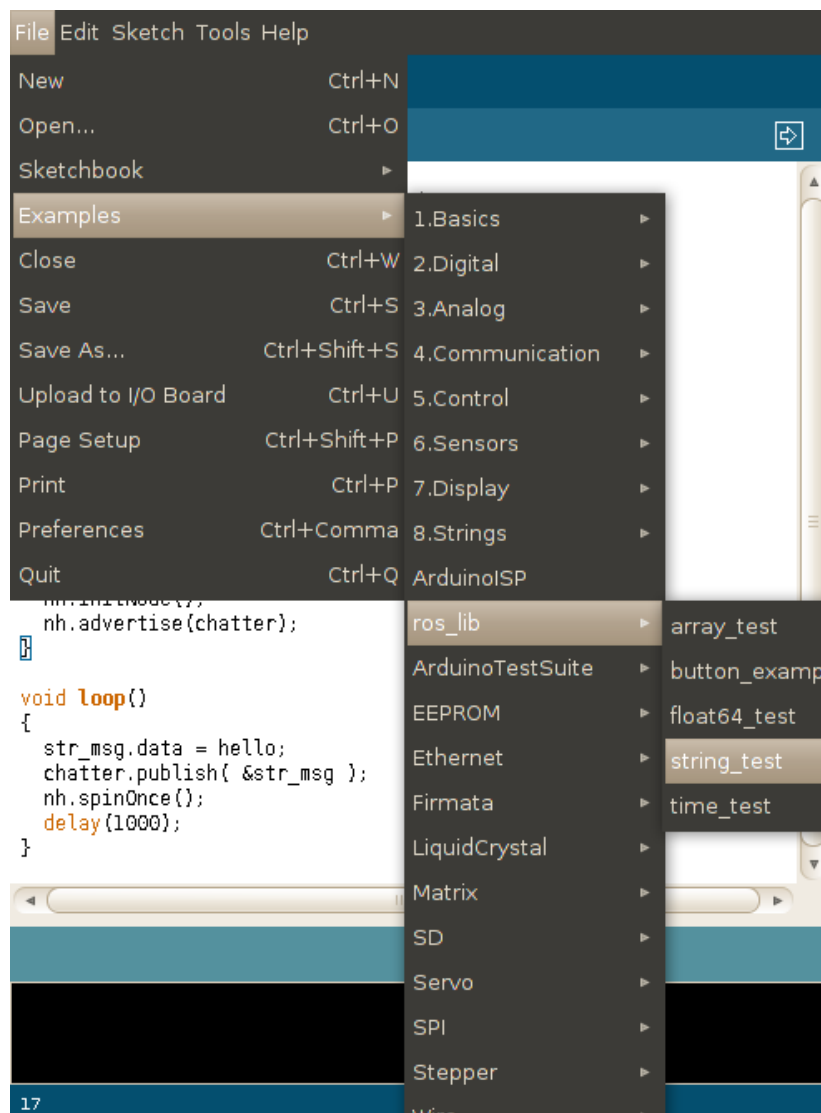


Figura 5. 1. Librería ROS

5.3. Agregar mensajes personalizados

En este apartado se muestra cómo generar archivos de encabezado del mensaje para el uso de los nuevos paquetes de mensajes con `rosserial` (es decir, cómo agregar el tipo de mensaje personalizado para su aplicación con `rosserial`).

El paquete `rosserial_client` incluye una herramienta para generar los archivos de cabecera necesarios de los archivos de definición de mensajes .

```
roslaunch rosserial_client path_to_libraries make_library.py your_message_package
```

en el caso de este proyecto se generan las cabeceras de las definiciones contenidas en el mensaje del paquete implementado ROS `interfaz_arduino` y `ros_lib` se encuentra en `'~/sketchbook/libraries/ros_lib'` por tanto el comando a utilizar es el siguiente

```
roslaunch rosserial_client make_library.py~/sketchbook/libraries interfaz_arduino
```

Este comando creará los encabezados, y los ubica en:

```
~/sketchbook/libraries/ros_lib/interfaz_arduino/.
```

Con esta instrucción se consigue generar los archivos de cabecera de mensajes importados desde el paquete ROS `interfaz_arduino` para su inclusión en el IDE Arduino.

5.4. Descripción de un programa de Arduino

Todo programa de Arduino contiene una función `void setup()` y una bucle infinito `void loop()`.

La función `setup()` se invoca una sola vez cuando el programa empieza. Se utiliza para inicializar los modos de trabajo de los pins, o el puerto serie. Debe ser incluido en un programa aunque no haya declaración que ejecutar.

Después de llamar a `setup()`, la función `loop()` hace precisamente lo que sugiere su nombre, se ejecuta de forma cíclica, lo que posibilita que el programa este respondiendo continuamente ante los eventos que se produzcan en la tarjeta .

El código está escrito mediante el IDE de Arduino y esencialmente consiste en la implementación de los topics para comunicar información a la interfaz gráfica Qt una vez leídos los puertos tanto digitales como analógicos y la implementación de los topics necesarios para cambiar el valor de las entradas digitales y analógicas registrados en la interfaz de usuario.

5.5. Flujograma

Este flujograma representa la ejecución del código de Arduino que recordemos permite ser considerado como nodo ROS mediante la librería roserial. Como se muestra en la figura 5.2 existe un *void setup ()* y un bucle infinito o *void loop ()*.

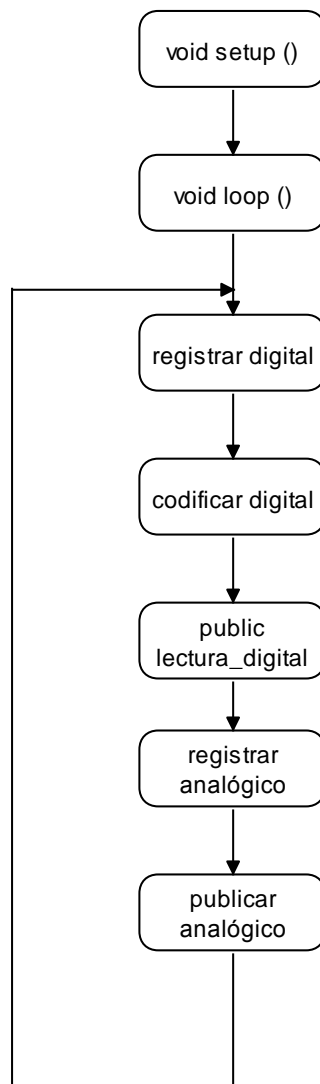


Figura 5. 2. Flujograma IDE-Arduino

A continuación se describen las distintas funciones:

- *Void setup()*: declaraciones de pines digitales correspondientes como salidas (del pin 22 al 38) y entradas (pin 39 al 53) así como la declaración de PWM como salidas pines digitales (del 0 al 13). Inicio de nodo ROS “roscnode_serial” así como la declaración para la ejecución de tanto subscriptores como publicadores.

- *Void loop()*: codificación en una variable de tipo entera de cada una de las 16 entradas digitales (39-53). Esta codificación se realiza mediante la comparación a nivel de bit de cada una de las entradas, posicionando un 0V, para un nivel de lectura bajo, o un 1.

Para un nivel de lectura alto 5V. Almacenando estos resultados empezando por el bit menos significativo en la variable entera de almacenamiento.

```
Int dato=0, Valor ,n=0; //inicialización de la variable almacenamiento y auxiliares (valor y n)

For(int j=38; j<53; j++){ //bucle for que recorre las 16 entradas

Valor=digitalRead(j); //lectura del pin

Dato=dato|(valor<<n); // almacenamiento de cada lectura

N++; //aumento de la variable de control n para almacenar el siguiente valor
```

Una vez registradas las 16 entradas analógicas codificadas en la variable dato publicamos este dato en el topic **lectura_digital**.

```
Digital_msg.data=valor; //mensaje digital_msg

Digital.publish(&digital_msg) //publicación ROS.
```

Para las entradas de tipo analógica el método es algo distinto, se ha creado un mensaje personalizado ROS ("pwm_msg), este incluye 16 variables de tipo entero una para cada entrada analógica de tal modo que el programa adquiere cada valor de estas entradas para almacenarlo en su correspondiente variable del mensaje personalizado:

```
Adc_msg.AD0 =analogRead(A0); // lectura de la primera entrada analógica.
```

Tras haber registrado todas la entradas (AD0, AD1.....AD15) estas son publicadas en ROS

```
Analógico.publish(&adc_msg); //con esto se publican las 16 variables pertenecientes al mensaje personalizado "adc"
```

De esta manera el *void loop* ejecuta continuamente una actualización de las todos los pines declarados como entradas para su posterior publicación en ROS.

El resto del código no esta incluido en este prototipo de programa de arduino y es el siguiente:

- **Declaraciones:** fuera de estas dos funciones ya comentadas *void setup()* y *void loop()*. El programa recoge la declaraciones necesarias de ROS (Nodo, publicadores y subscriptores y tipos de mensajes).

```
Ros::NodeHandle nh; //inicio Nodo

Ros::Subscriber<std_msgs::Int16> sub2("pwm",&pwm); //subscriber pwm

Ros::Subscriber<std_msgs::Int16> sub( "entradas_digitales",&configuracion);
//subscriber digital

Ros::Publisher digital ("lectura_digital",&digital_msg); //publicador digital
```

```
Ros::Publisher analógico ("lectura_analógica",&adc_msg); //publicador analógico.
```

- **Funciones:** Existen dos funciones una para cada subscritor las cuales reciben los datos necesarios para escribir en las salidas tanto digitales como PWM.

La primera función es pin y su flujograma se muestra en la figura 5.3

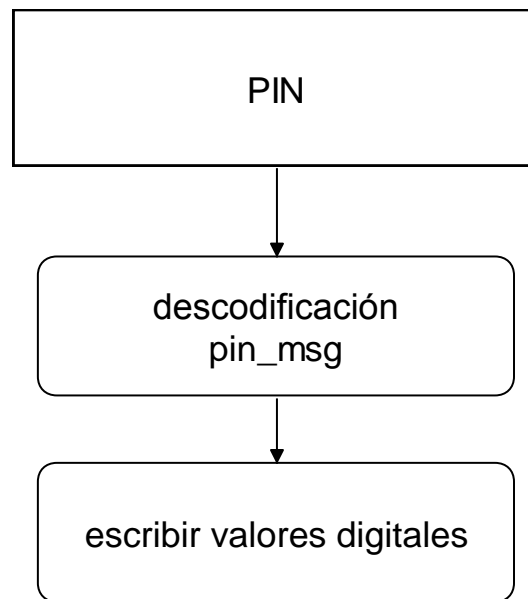


Figura 5. 3. Función PIN

Recibe la información codificada a nivel de bit de cada una de las 16 salidas digitales para después descomprimirla y escribir los valores digitales altos o bajos según corresponda.

```

Void configuración (const std_msgs::Int16& cmd_msg){
    Int aux=1; //variable de control
    For(int i=22; i<38; i++){
        If ((cmd_msg.data & aux)==aux){ //comparación a nivel bit
            digitalWrite(l, HIGH); // activar salida 5V.
        }
        else{
            digitalWrite(l,LOW); // desactivar salida 0V.
        }
        aux=aux*2; //siguiente salida a analizar
    }
}
  
```


La segunda función es la pwm y su flujograma se muestra en la figura 5.4.

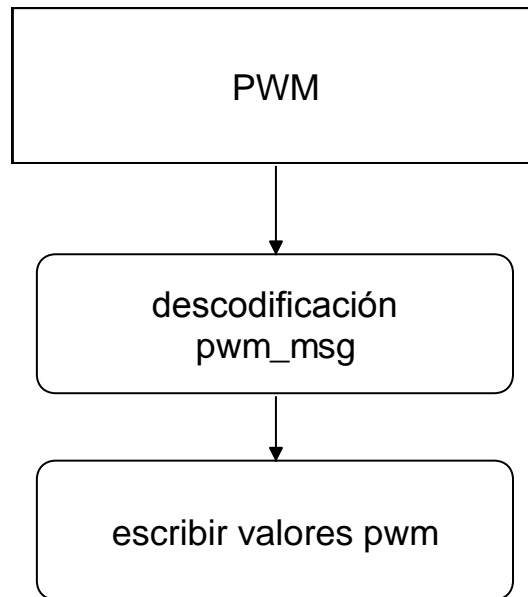


Figura 5. 4. Función PWM

Se recibe el mensaje pwm desde la interfaz_arduino formado por 12 enteros correspondientes a cada uno de los pines pwm. La función se encarga de escribir estos valores en las correspondientes pwm variando su ciclo de trabajo.



Capítulo 6: IMPLEMENTACIÓN

ROS-QT

6.1 Creación proyecto

Para la implementación de ROS-QT se ha instanciado el código en un paquete normal Ros modificando posteriormente su *CmakeList.txt* para introducir las instrucciones necesarias para la generación del código QT.

Dentro de este proyecto se generan todos los archivos necesarios, tanto los archivos de cabecera de **extensión.h**, los archivos fuente de **extensión.cpp** y los archivos autogenerados de mensajes personalizados.

6.1.1. Creación paquete ROS

Una vez configuradas y exportadas las variables de entorno ROS en la carpeta de trabajo **ros_workspace** utilizamos la plantilla **roscpp_create_pkg** [nombre de paquete][dependencias] instanciado en la mencionada carpeta de trabajo. Estas dependencias serán *std_msgs roscpp* (cliente c++) y *rospy* (cliente python para futuras ampliaciones).

De esta manera usando el comando **roscpp_create_pkg interfaz_arduino std_msgs roscpp rospy** creamos el paquete ROS de nombre **interfaz_arduino** con las dependencias necesarias.

6.1.2 Creación mensajes personalizados ROS

Para terminar el paquete ROS creamos los mensajes personalizados de tipo MSG. Los MSG son simples archivos de texto que describen los campos de un mensaje de ROS. Se utilizan para generar código fuente para mensajes en diferentes lenguajes. Los MSG se almacenan en el MSG directorio de un paquete.

6.1.3. Modificación cMakelist.txt

Una vez generado el paquete Ros debemos indicar al compilador las instrucciones necesarias para incluir código QT. De esta manera cuándo ejecutemos el comando *make* en Linux, cuyo propósito es determinar automáticamente qué piezas de un programa necesitan ser recompiladas y lanzar las órdenes para recompilarlas, se creará el ejecutable que incluirá un nodo ROS y el código propio QT para la representación de la interfaz de usuario. La **cMakeList.txt** se muestra a continuación:

```
cmake_minimum_required(VERSION 2.4.6)
include($ENV{ROS_ROOT}/core/rosbuild/rosbuild.cmake) //commando make de ROS

# Set the build type. Options are:
# Coverage      : w/ debug symbols, w/o optimization, w/ code-coverage
# Debug         : w/ debug symbols, w/o optimization
# Release       : w/o debug symbols, w/ optimization
# RelWithDebInfo : w/ debug symbols, w/ optimization
# MinSizeRel    : w/o debug symbols, w/ optimization, stripped binaries
#set(ROS_BUILD_TYPE RelWithDebInfo)

rosbuild_init()

find_package(Qt4 REQUIRED)
# enable/disable some Qt features
set( QT_USE_QTGUI TRUE )
set( QT_USE_QTOPENGL TRUE )
set( QT_USE_QTXML TRUE )
include(${QT_USE_FILE})

ADD_DEFINITIONS()

set(qt_srcs    //archivos.cpp incluidos en el proyecto

    src/arduino.cpp
    src/pin.cpp
    src/nodo.cpp
    src/pwm.cpp
)
set(qt_hdrs //archivos.h
    src/pin.h
    src/arduino.h
    src/nodo.h
    src/pwm.h
)

QT4_ADD_RESOURCES(QT_RESOURCES_CPP ${QT_RESOURCES}) //instrucciones
necesarias para incluir las archivos MOC QT autogenerados
qt4_automoc(${qt_srcs})
QT4_WRAP_CPP(qt_moc_srcs ${qt_hdrs})
```

```
# include this for ui_h
include_directories(${CMAKE_CURRENT_BINARY_DIR})
rosbuild_add_executable(interfaz_arduino src/main.cpp    //compilar ejecutable en
main.cpp

    ${uis_h} ${qt_srcs} ${qt_moc_srcs} ${QT_RESOURCES})
target_link_libraries(interfaz_arduino ${QT_LIBRARIES})

#set the default path for built executables to the "bin" directory
set(EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
#set the default path for built libraries to the "lib" directory


#uncomment if you have defined messages
rosbuild_genmsg() //inclusión de archivos personalizados ROS
#uncomment if you have defined services
#rosbuild_gensrv()

#common commands for building c++ executables and libraries
#rosbuild_add_library(${PROJECT_NAME} src/example.cpp)
#target_link_libraries(${PROJECT_NAME} another_library)
#rosbuild_add_boost_directories()
#rosbuild_link_boost(${PROJECT_NAME} thread)
#rosbuild_add_executable(example examples/example.cpp)
#target_link_libraries(example ${PROJECT_NAME})
```

6.2. Flujograma principal

El flujograma sobre la relación de los diferentes archivos instanciados en el paquete anteriormente mencionado de ROS se muestra en la figura 6.1. Estos archivos corresponden a las diferentes clases del programa girando sobre su clase principal Arduino y las subclases de éste que son nodo, pin y pwm.

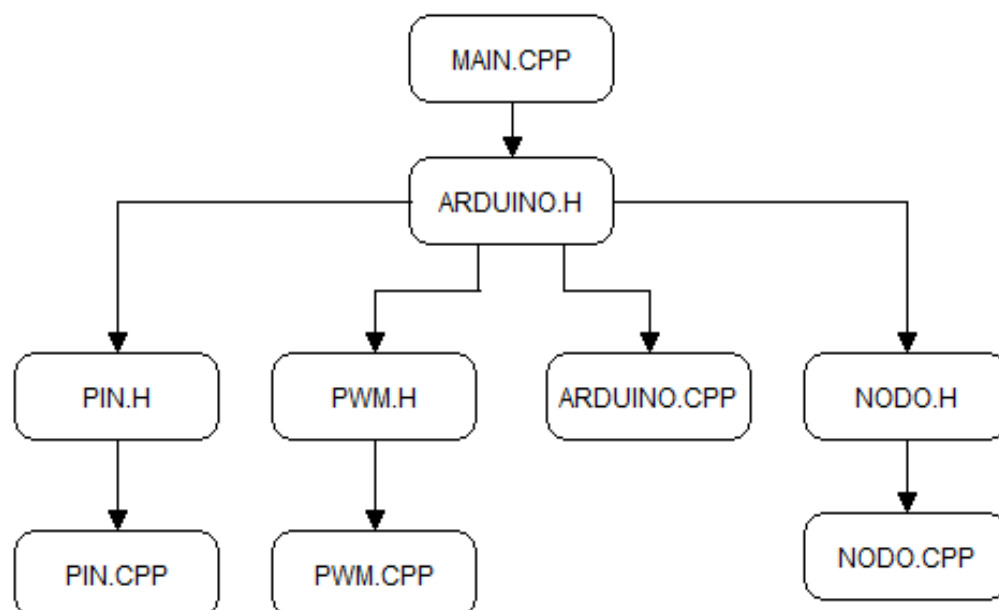


Figura 6. 1. interacción de archivos

Este flujograma representa la interacción entre los diferentes archivos del programa. De este modo el **main.cpp** recoge las instrucciones para la ejecución de la aplicación QT, instanciando una clase Arduino que recoge todas las demás clases de la aplicación y donde se encuentra todo el código funcional.

Tras haber ejecutado el programa recogido en **main.cpp** el proceso es el siguiente:

- Creación clase Arduino que recoge todo el código funcional.
- La clase Arduino crea toda la funcionalidad de la aplicación a través de sus clases (pin,pwm y nodo).
- En **main.cpp** se muestra la clase Arduino mediante el comando *show*.
- Se ejecuta la aplicación.

6.3. Clase principal Arduino

Como se ha visto en el flujograma anterior de la figura 5.4, la clase principal es Arduino. Sobre esta clase está escrito todo el código de implementación QT ya sea mediante clases intrínsecas en QT o las subclases creadas.

6.3.1. Flujograma Arduino

El flujograma de la clase Arduino se muestra en la figura 5.5

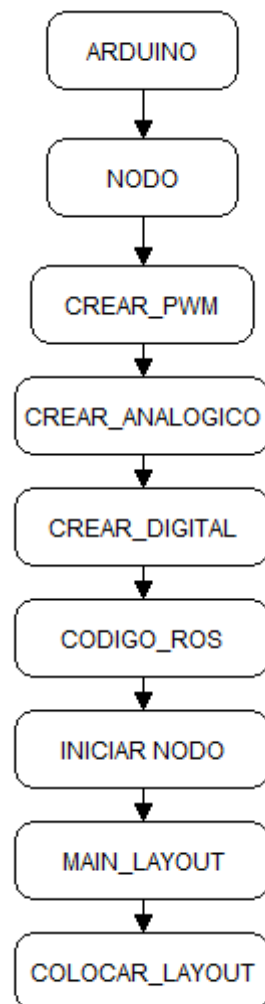


Figura 6. 2. Flujograma clase principal

Al iniciar la clase Arduino desde el archivo ejecutable **main.cpp** se pasan como argumentos **int**, **arg** y **argv** o argumentos de línea de comandos para poder iniciar el nodo. Tras esto se llaman a las funciones *void crear()*, encargadas de crear los diferentes widgets (PWM, analógico y pin).

Cada una de estas funciones `void crear()` generan los widgets elegidos para representar los pines de la placa. Ambas siguen el mismo protocolo instanciando un cuadro de dialogo o `QGroupBox` con su título correspondiente. En este `QGroupBox` se crea una `QGridLayout` y se posicionan los widgets siguiendo una distribución de filas y columnas.

En lo referente al código ROS se crean los publicadores y mensajes necesarios para la interacción con Arduino, los subscriptores que necesitan actualizarse constantemente han sido declarados en el archivo `Nodo` consistente en un `QThread` cuya característica esencial es una ejecución en paralelo con el programa principal.

Una vez generados los widgets y el código ROS necesario se implementa las interacciones entre los diferentes archivos por medio de señales que son emitidas desde una de las subclases y recibidas en la clase principal `Arduino`.

Por último, desarrollada ya la funcionalidad del programa, colocamos los diferentes cuadros de diálogos en otro `QGridLayout` pero está posicionándolos según columnas para homogenizar el diseño.

6.3.2 VOID CREAR ANALOGICO ():

Función en la cual se crean los widgets necesarios para la implementación de las entradas como se muestra en la figura 6.3

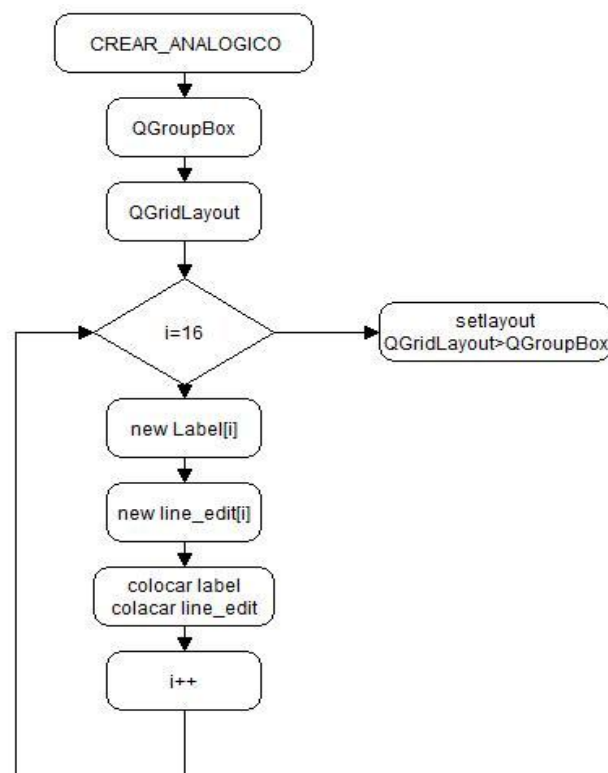


Figura 6. 3. Flujograma de la creación y colocación de entradas analógicas

Para representar las entradas analógicas de la placa se ha elegido la clase `QLineEdit` junto con una etiqueta representada por un `QLabel`.

La clase `QLineEdit` situada en el marco QT representa una línea de edición de texto que puede ser leída y escrita. Para nuestro caso sólo necesitamos que sea escrita con los valores de las entradas analógicas. Esta clase tiene unas propiedades y funciones propias para controlar el diseño, la escritura, la lectura etc. para nuestro caso sólo ha utilizado la función de escritura que comentaremos posteriormente.

En cuanto a la clase `QLabel` implementa una etiqueta con el nombre `A%argumento`, siendo este el número de `QLineEdit` correspondiente desde la 0 a la 15.

Esta función `void crear()` se encarga de implementar dichos widgets. Primero se crea un cuadro de dialogo o `QGroupBox` con el nombre correspondiente a las objetos que va a albergar en este caso ("PANEL ANALOGICO"). Un bucle `for` es el encargado de crear cada uno de los labels o etiquetas con el nombre apropiado y una `lineEdit` (o línea de texto) estos dos widgets son colocados en un `QGridLayout` siguiendo una distribución de filas y columnas instanciando la etiqueta[i] en la fila i columna 0 y la línea de texto[i] en la misma fila pero en la columna 1. Obteniendo una ordenación vertical. Una vez recorrido el bucle y completadas las 16 entradas analógicas el `QGridLayout` es introducido en la cuadro dialogo previamente creado.

Un ejemplo del diseño obtenido se muestra en la figura 6.4

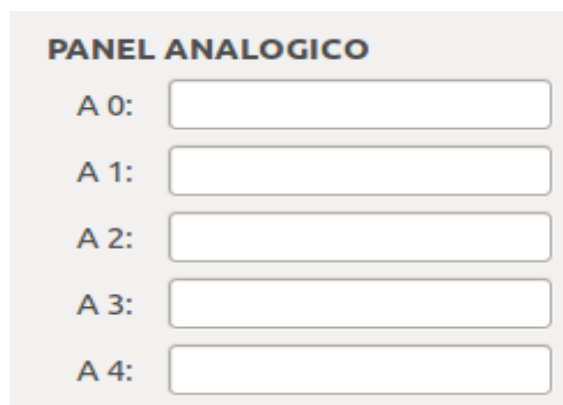


Figura 6. 4. Representación de las entradas analógicas

6.3.3. VOID CREAR PWM ():

La función pwm crea los widgets necesarios para representar las pwm y su colocación en forma de panel.

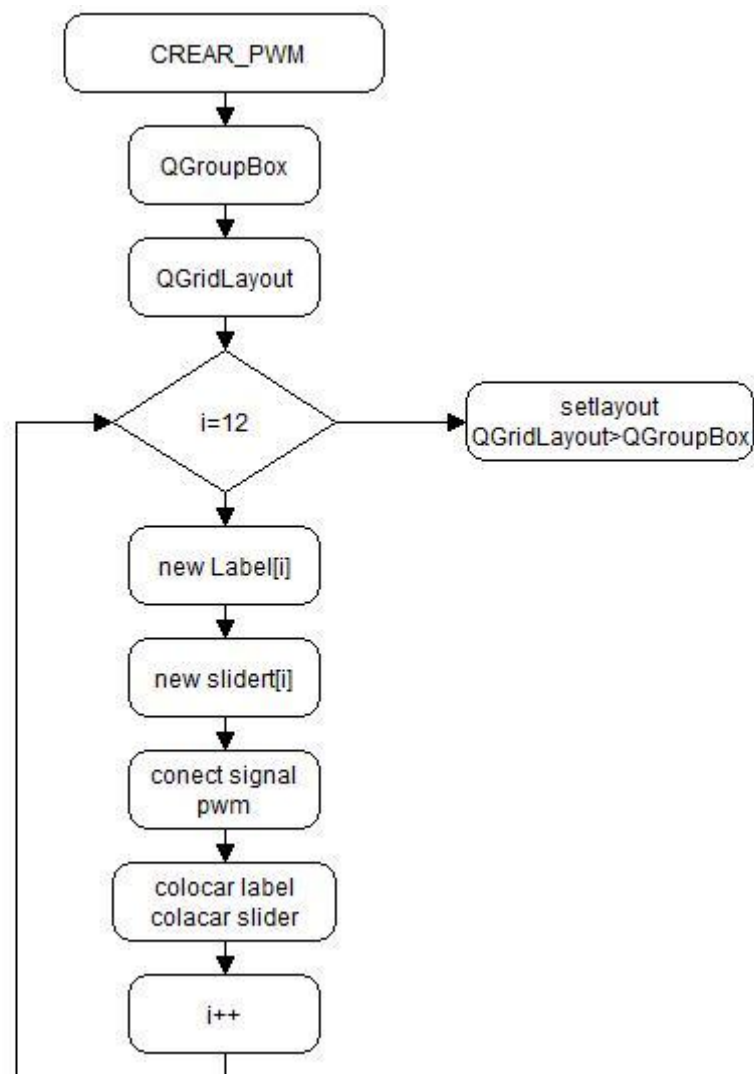


Figura 6. 5. Flujograma de la creación PWM

En el caso de las PWM se ha elegido un QSlider, que consiste en un control deslizante, en este caso horizontal para controlar el ciclo de trabajo de las PWM que hemos fijado en porcentaje, estableciendo un intervalo máximo que corresponde con 100.

El funcionamiento de la función *void crear_pwm* es similar a *void Crear_analogico()*, es la encargada de crear los widgets con sus etiquetas que corresponden con cada una de las salidas PWM de la placa. También se han implementado la conexión de señales de cada PWM, de esta manera cada vez que el valor de alguna de éstas es modificado se envía una señal que permite acceder a la función correspondiente para publicar en el tema de Ros "PWM".

Un ejemplo del diseño obtenido se muestra en la figura 6.6

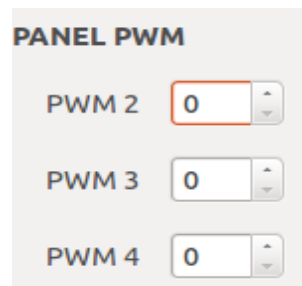


Figura 6. 6. Representación de las PWM

6.3.4. VOID CREAR DIGITAL ():

La función *crear digital* crea los *QPushButton* de representación de los pines digitales de la placa, su flujograma corresponde al de la figura 6.7

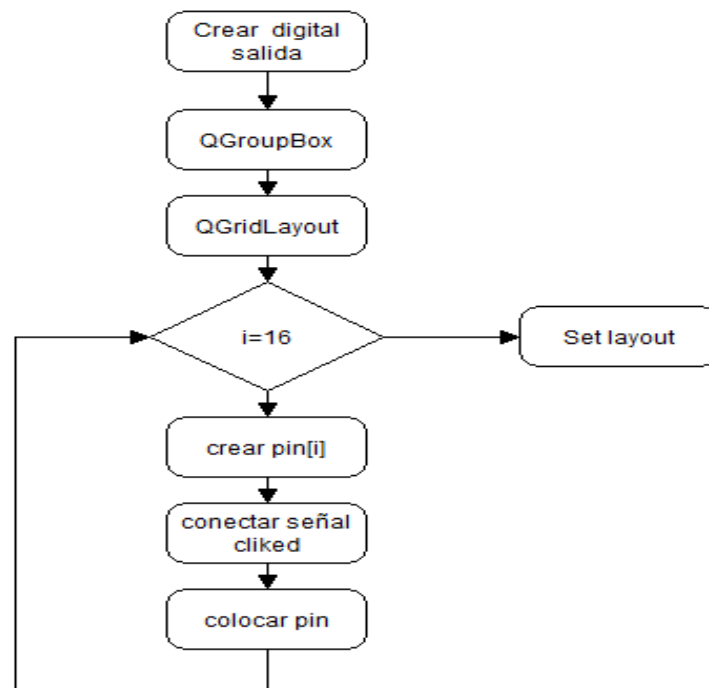


Figura 6. 7. Flujograma de la función *crear_digital*

Crear digital es homólogo a las anteriores funciones, en este caso el nombre del QGroupBox es PANEL SALIDAS DIGITALES. Se ha elegido como widgets de representación de estos pines digitales un QPushButton implementado en la clase **pin.cpp**. Al recorrer el bucle *for* creando los 16 botones también se conectan las señales necesarias para registrar los cambios producidos en estos y poder publicarlos en el tema ROS “entradas_digitales”. La colocación de los botones se realiza en un nuevo QGridLayout con una distribución de 8 filas y 2 columnas.

En la figura 6.8 se muestra el diseño de las entradas digitales y en la 6.9 el de las salidas digitales



Figura 6. 8. Entradas digitales



Figura 6. 9. Salidas digitales

Se realiza otro cuadro de dialogo o GroupBox con nombre PANEL ENTRADAS DIGITALES, en el que se implementan los restantes 16 botones correspondiendo a las 16 entradas digitales con la misma colocación pero sin conectar ninguna señal ya que estos botones son de lectura.

Tras la creación de todos los widgets de representación de pines de Arduino se procede a la inicialización del código de ROS incluyendo la información de los publicadores, ya que son estos los que envían la información cuando se producen cambios en los botones del panel de entradas analógicas y en el panel de PWM. También se incluyen los mensajes utilizados para dichos publicadores y algunas instrucciones más para permitir el uso del nodo fuera del scope.

Por último, necesitamos colocar estos cuadros de dialogo en un mainLayout principal colocado verticalmente y siguiendo este orden (PANEL PWM, PANEL ANALOGICO, PANEL ENTRADAS DIGITALES Y PANEL SALIDAS DIGITALES).

6.3 Clases utilizadas

Se han creado 3 clases para implementar la funcionalidad del programa estas son pin, PWM y nodo. Cada una de estas clases dispone de un archivo de cabeza con extensión .h y un archivo fuente de extensión .cpp

6.3.1 PIN

Describe la funcionalidad de cada uno de los pines digitales implementados. Este es el flujograma del archivo **pin.cpp** ya que su cabecera sí incluye la definición de clase, signals y slot. Se muestra en la figura 6.10

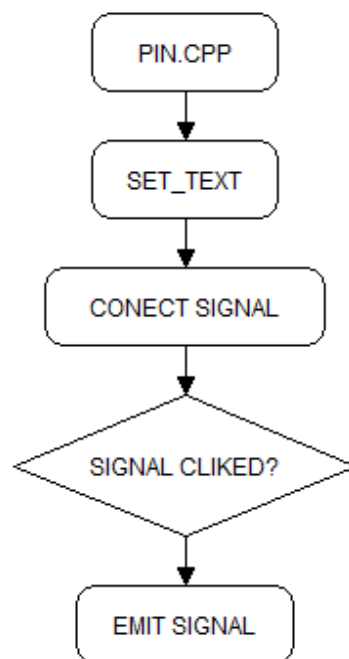


Figura 6. 10. Flujograma PIN

La clase pin es utilizada para crear los botones de las entradas y salidas digitales, se ha utilizado esta clase para crear cada uno de los botones con su identificador, siendo este el número de botón. Desde la función *crear_digital ()* ya comentada anteriormente se crea cada uno de los pines pasando como argumento el número de QPushButton implementado, este argumento se utiliza para colocar el texto correspondiente dentro del botón desde el numero 22 al 53. También se implementa la señal **clicked** que se emite una vez hemos pulsado el botón, emitiendo como consecuencia una nueva señal con el número de botón que hemos pulsado.

6.3.2 PWM

Describe la funcionalidad de cada uno de los pines digitales implementados como pwm. Este es el flujograma del archivo **pwm.cpp** ya que su cabecera si incluye la definición de clase, signals y slot. Se muestra en la figura 6.11

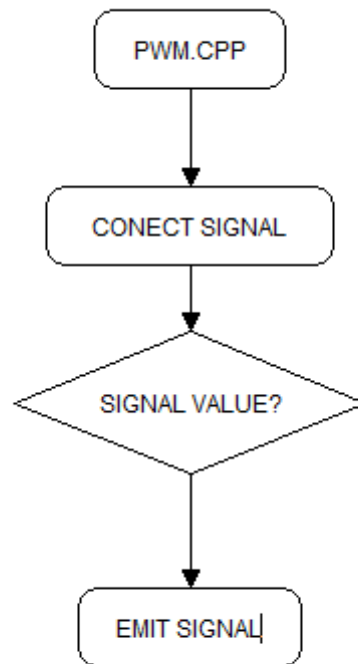


Figura 6. 11. Flujograma PWM

La clase PWM sigue el mismo funcionamiento que pin. En este caso la señal conectada es *value*, que se emite cuando el valor de QSpinBox es modificado, y emitiendo a su vez una nueva señal pasando como argumentos el valor modificado (ciclo de trabajo) y el identificador de la PWM (de 0 a 12).

6.3.3 NODO

Esta clase no implementa ningún widget sino que describe la comunicación con ROS. Es un proceso que se ejecuta en paralelo con la clase principal. Ver figura 6.12

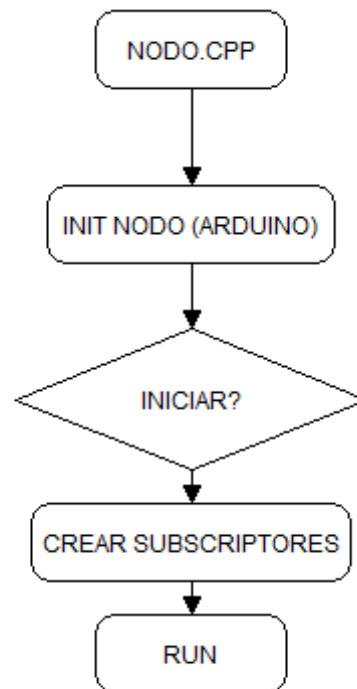


Figura 6. 12. Flujograma clase NODO

La clase nodo es la encargada de implementar toda la comunicación ROS de suscriptores. Se ha implementado en un QThread, objeto QT, el cual tiene como característica principal una ejecución en paralelo con el programa principal. Este tipo de clase tiene un funcionamiento específico y que principalmente consiste en una función que inicia el Qthread y el código que se ejecuta mientras este activo en la función RUN.

En este caso la clase nodo implementa el `Ros::init` para crear el nodo necesario con nombre arduino. Desde el archivo `Arduino.cpp` es llamada la función `iniciar` momento en el cual empieza la ejecución del QThread creando además los suscriptores con topics "lectura_digital" y "lectura_analógica" que ejecutan las funciones correspondientes cada vez que se reciben datos en dichos topics

En la función `run` solo es necesario ejecutar estos suscriptores continuamente con la función intrínseca en `ros.h` `ROS::spin()`;

6.4. Implementación funcionalidad: SIGNALS Y SLOT

En la figura 6.13 se muestra el funcionamiento en cuanto archivos signals y slot de los pines digitales.

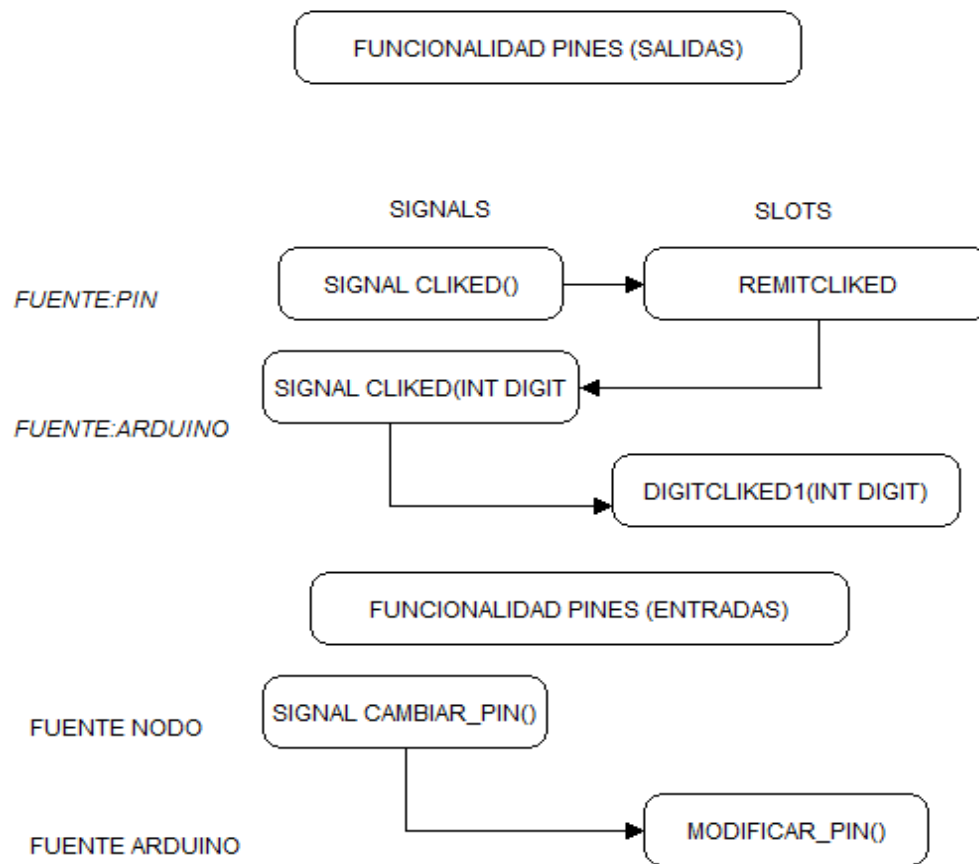


Figura 6. 13. Funcionalidad PIN

El funcionamiento de las salidas digitales que como ya hemos mencionado antes son botones es el siguiente: desde el archivo **pin.cpp** se emite la señal `cliked` cuando pinchamos el botón izquierdo del ratón sobre dicho botón. Esta señal inicia la función o slot `remit_cliked` que emite una nueva señal `cliked` pero con el identificador del botón pulsado. Desde el archivo **arduino.cpp** esta señal es conectada al slot `digitcliked` pasando como argumento el identificador, esta función es la encargada de registrar y publicar los cambios en el topic "escritura_digital":

- *Función digitclicked1(int digit):*

Almacena los cambios registrados en cada botón almacenando un valor de 1 (5V.) o 0 (0V.) en un array llamado valores de subíndice 16. Según estos valores además la función se encarga de cambiar el color del botón ya sea verde para 5V o rojo para 0V. Por último se llama a la función calcular para registrar a nivel de bit cada uno de los valores del array para posteriormente publicarlo en ROS.

Se publicarán nuevos datos cada vez que pulsemos sobre cualquier botón catalogado como salida mediante la función *digitclicked1*.

El funcionamiento de las entradas digitales es el siguiente: cuando un nuevo mensaje se publique en “lectura_digital” el archivo nodo.cpp emitirá una señal activando el slot modificar pin que se encargará de cambiar el color de cada botón según sea verde o rojo.

En la figura 6.14 se muestra la funcionalidad de las entradas analógicas.

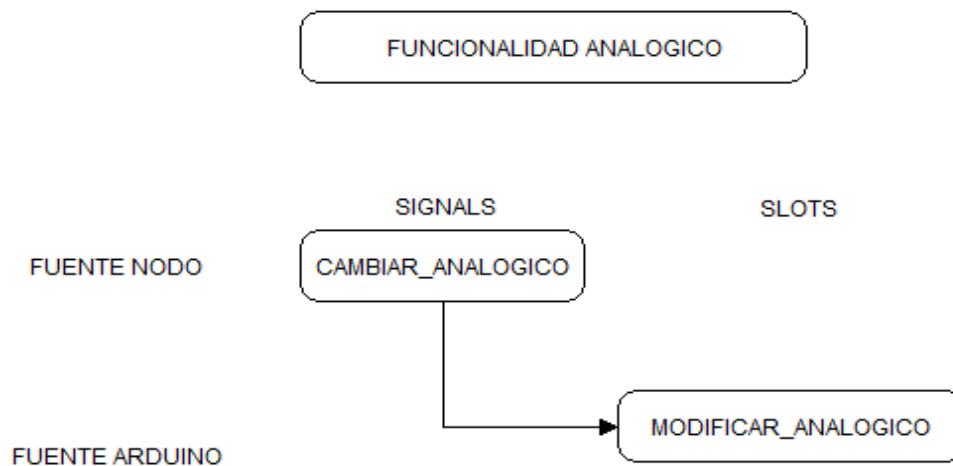


Figura 6. 14. Funcionalidad analógico

Para la funcionalidad analógica cuando se publican nuevos mensajes en el topic “lectura analógica” la función analógico implementada en el archivo nodo almacena los datos del mensaje y después emite estos datos en la señal cambiar analógico. Desde el archivo Arduino la señal cambiar analógico activa el slot modificar analógico que se encarga de escribir los valores correspondientes en cada line_edit según los datos obtenidos.

En la figura 6.15 se muestra la funcionalidad de las salidas PWM.

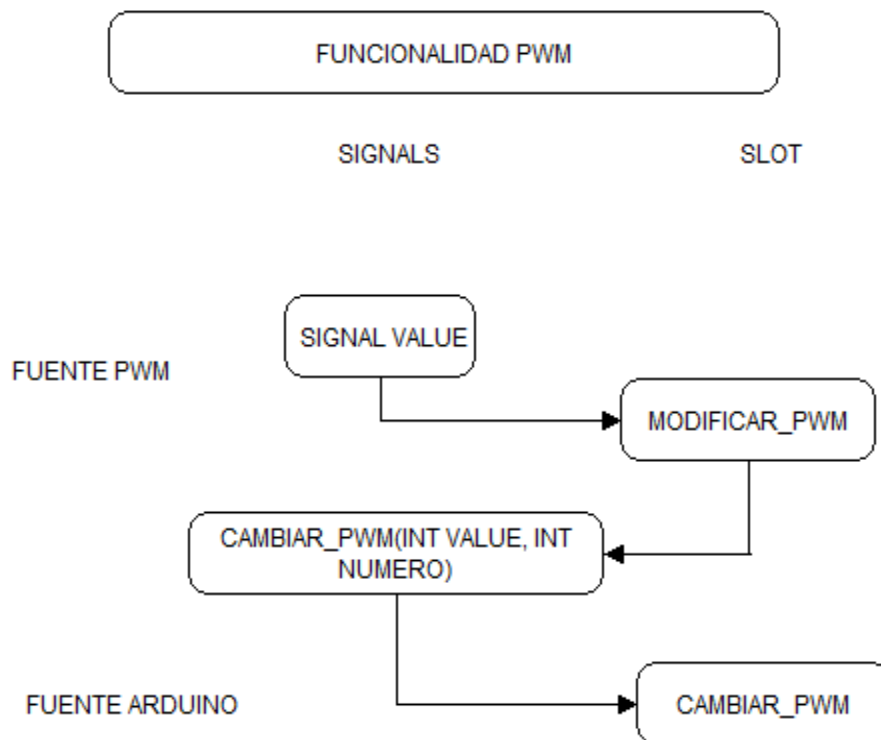


Figura 6. 15. Funcionalidad PWM

La señal value es emitida cuando cambiamos el valor de la pwm, se trata de una señal intrínseca a la clase QSpinBox de las que son tipo las pwm. Emitiendo esta señal accedemos a la función o slot `modificar_pwm` en el archivo `pwm.cpp` que emite una nueva señal con el valor cambiado y el identificador de pwm correspondiente (del 0 al 12 según la pwm de la cual cambiamos el valor). Desde el archivo `arduino.cpp` registramos esta señal y ejecutamos el slot `cambiar_pwm` que se encarga de almacenar los valores de 0-100 de cada pwm. Después estos se publican en el topic `pwm`.

Capítulo 7: RESULTADOS

EXPERIMENTALES

7.1. Introducción.

En este capítulo se detallan los resultados obtenidos con el montaje de varios circuitos sencillos para manipular las entradas y salidas digitales y obtener de forma visual los resultados.

7.2. Inicialización.

Para inicializar tanto la interfaz de usuario como el rosnodeSerial necesitamos iniciar ROS mediante el comando de línea roscore. Tras la ejecución del comando se inicializa el ROS master que permite la comunicación y ejecución de todos los nodos ROS. Ver figura 7.1

```
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:38008/
ros_comm version 1.6.6

SUMMARY
=====

PARAMETERS
* /rosversion
* /rostdistro

NODES

auto-starting new master
process[master]: started with pid [7139]
ROS_MASTER_URI=http://ubuntu:11311/

setting /run_id to 6a57d2fc-2826-11e2-af87-4c0f6e5c590b
process[rosout-1]: started with pid [7152]
started core service [/rosout]
```

Figura 7. 1. Inicialización roscore

Encontrándose el maestro ROS inicializado podemos ejecutar cualquier código que incluya un nodo ROS. Por tanto podemos ejecutar indistintamente cualquiera de los nodos utilizados en el programa

7.2.1. Ejecución SERIALNODE

Inicia el nodo atribuido a la placa Arduino especificando el puerto de conexión, mediante el siguiente comando `roslaunch rosserial_python serial.node.py /dev/ttyACM0` la última expresión corresponde con el puerto de conexión de Arduino. Ver figura 7.2

```
usuario@ubuntu:~$ roslaunch rosserial_python serial.node.py /dev/ttyACM0
[INFO] [WallTime: 1352215811.483829] ROS Serial Python Node
[INFO] [WallTime: 1352215811.486152] Connected on /dev/ttyACM0 at 57600 baud
[INFO] [WallTime: 1352215813.614007] Note: publish buffer size is 512 bytes
[INFO] [WallTime: 1352215813.614496] Setup publisher on lectura_digital [std_msgs/Int16]
[INFO] [WallTime: 1352215813.647432] Setup publisher on lectura_analogica [interfaz_arduino/adc]
[INFO] [WallTime: 1352215813.651465] Note: subscribe buffer size is 512 bytes
[INFO] [WallTime: 1352215813.651774] Setup subscriber on entradas_digitales [std_msgs/Int16]
[INFO] [WallTime: 1352215813.658529] Setup subscriber on pwm [interfaz_arduino/pwm]
```

Figura 7. 2. Inicialización serial/nodo

Como información tras la ejecución del comando se incluyen los publicadores y suscriptores del nodo, el tipo de mensaje del nodo y el tamaño del búfer así como la velocidad en baudios.

7.2.2. Ejecución INTERFAZ_ARDUINO

Ejecutamos la aplicación mediante el comando rosrún. Ver figura 7.3

```
usuario@ubuntu:~/ros_workspace/interfaz_arduino$ rosrún interfaz_arduino interfaz_arduino

(interfaz_arduino:7716): Gtk-WARNING **: Imposible encontrar el motor de temas e
n la ruta al _modulo: «pixmap»,

(interfaz_arduino:7716): Gtk-WARNING **: Imposible encontrar el motor de temas e
n la ruta al _modulo: «pixmap»,

(interfaz_arduino:7716): Gtk-WARNING **: Imposible encontrar el motor de temas e
n la ruta al _modulo: «pixmap»,

(interfaz_arduino:7716): Gtk-WARNING **: Imposible encontrar el motor de temas e
n la ruta al _modulo: «pixmap»,
```

Figura 7. 3. Inicialización de interfaz

Obteniendo la ventana de aplicación interfaz_arduino como se muestra en la figura 7.4

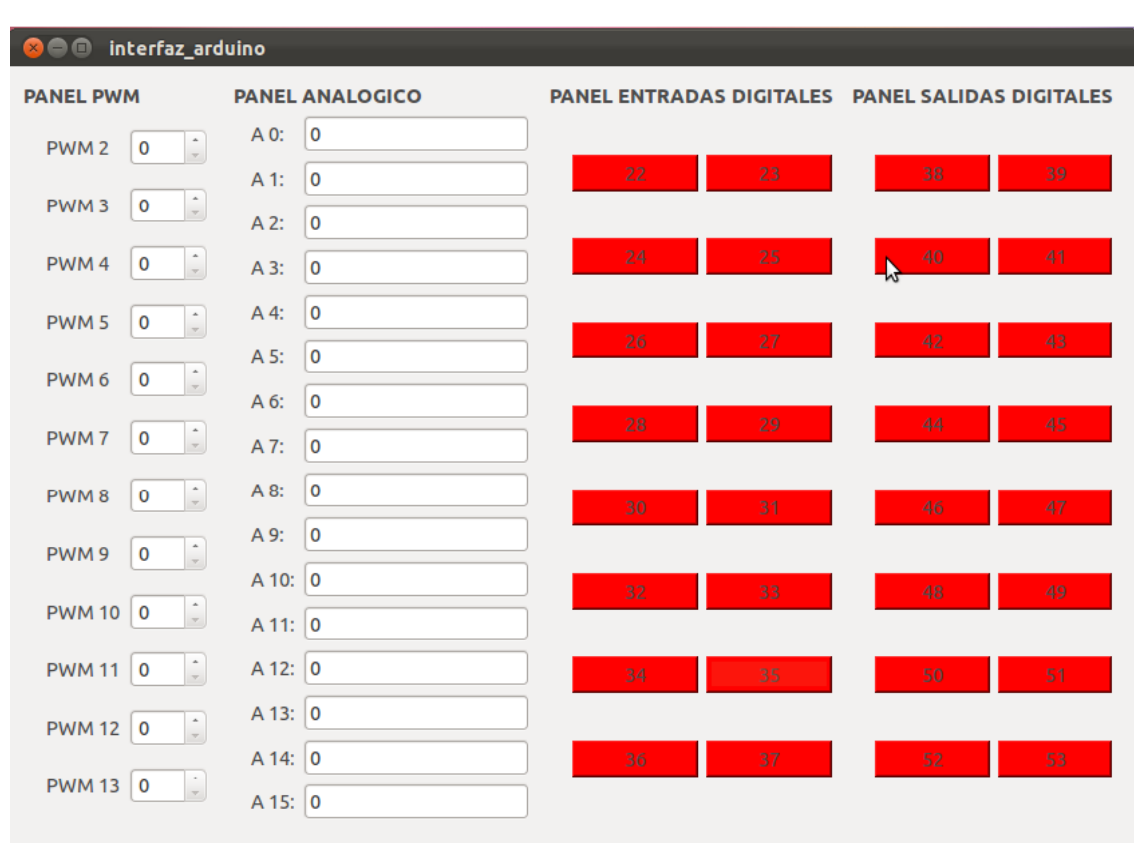


Figura 7. 4. Aplicación interfaz_arduino

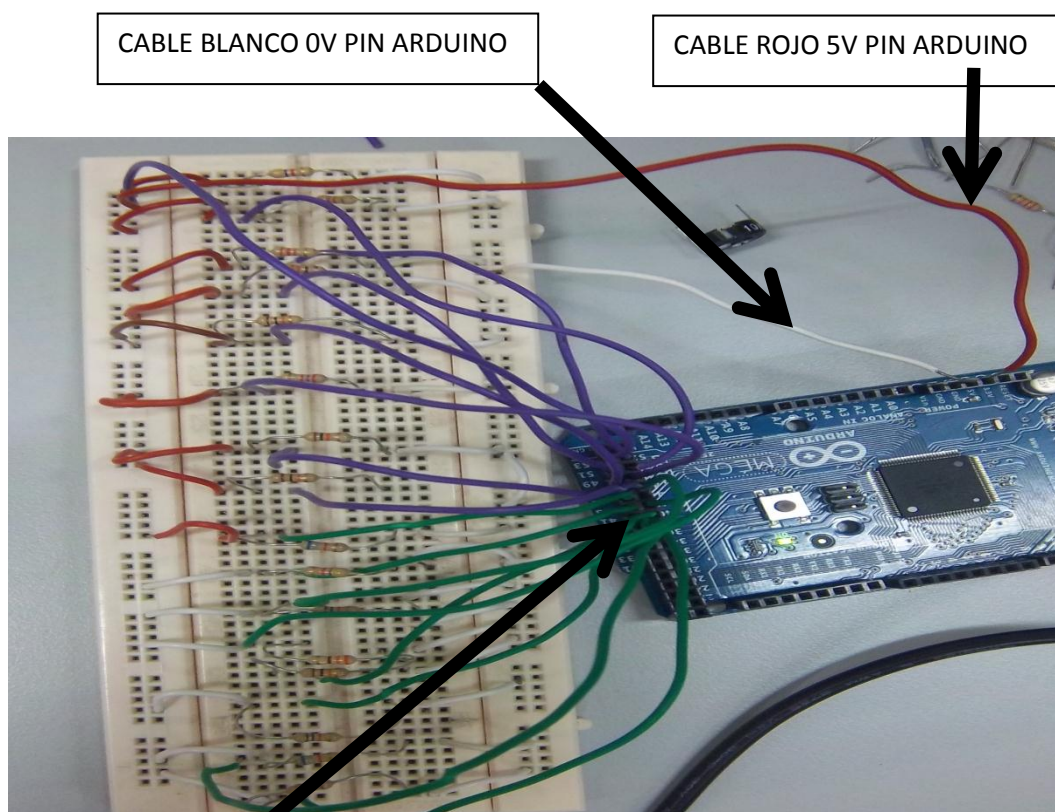
Todos los botones se muestran en rojo ya que las salidas son conectadas forzosamente a 0V como inicialización y las entradas recogen todas una lectura de nivel bajo o 0V.

7.3. Salidas digitales.

Se ha realizado un circuito conectando cada uno de los pines de entrada digital de la placa (del pin 38 al pin 53 ambos inclusive) a una resistencia para evitar interferencias entre estos. Alimentando las resistencias con 5V. El pin correspondiente registrara la lectura y el botón que representa cambiará su color a verde en caso contrario la lectura será de 0V y el color del botón será rojo.

7.3.1. Primer caso

Se han conectado la mitad de las entradas digitales a nivel alto lógico 5. Cambiando el color de rojo a verde de las entradas correspondientes. Como se muestra en la figura 7.5, el cable rojo corresponde con el pin 5V. De la placa Arduino obteniendo lecturas positivas en las entradas a las que está conectado esta tensión que corresponden con los últimos 8 pines. Los 8 primeros pin no están conectados a ninguna voltaje por lo que registran 0V.



CONEXIONES A LA PLACA

Figura 7. 5. Primer montaje de entradas digitales

En la figura 7.6 se muestran los resultados obtenidos en la interfaz para el montaje de la figura anterior 7.5. Se observa que los pines conectados a 5 V (los 8 últimos) se encuentran en verde mientras que los 8 primeros conectados a 0V están en rojo

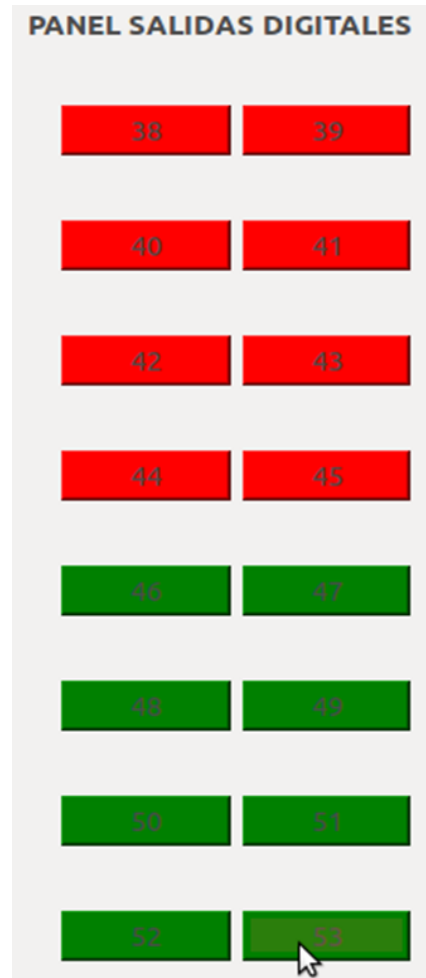


Figura 7. 6. Panel salidas digitales

7.3.2. Segundo caso

Para este segundo montaje figura 7.7 se han intercambiado las tensiones de los 8 primeros pines con los 8 segundos pines. Obteniendo los primeros una lectura de 5V. (Cable rojo desde Arduino en la figura 7.7) y los segundos una tensión de 0V.

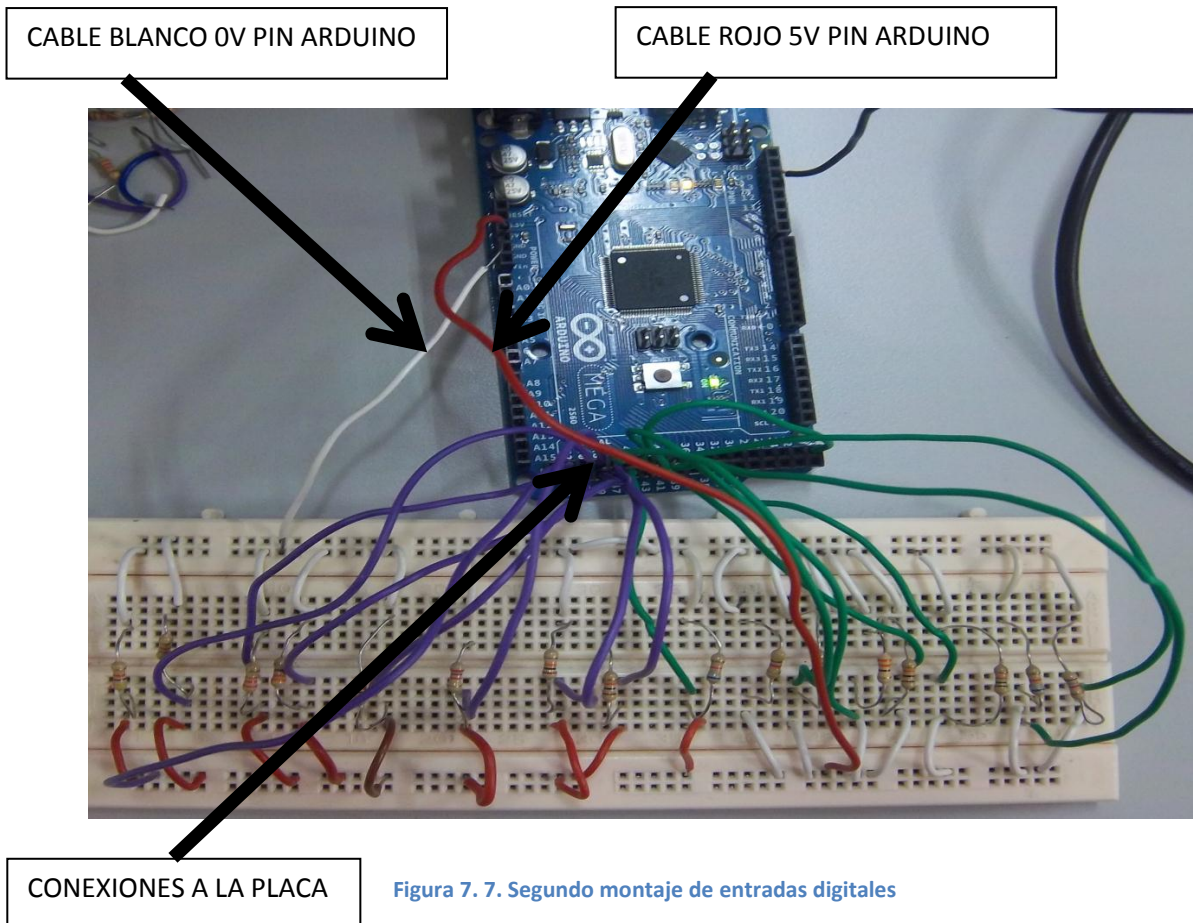


Figura 7. 7. Segundo montaje de entradas digitales

En la figura 7.8 se muestran los resultados en la aplicación para este segundo montaje.

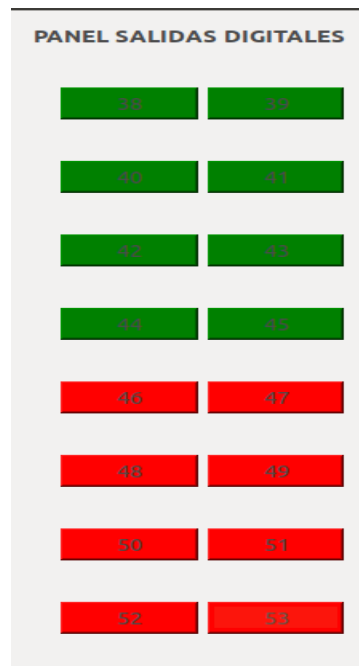


Figura 7. 8. Panel salidas digitales

7.4. Entradas digitales.

Se han conectado cada una de salidas digitales (pin 22 al 38) con las entradas analógicas (de la A0 a la A15) de esta manera comprobamos el funcionamiento de dos elementos de la interfaz gráfica. El valor escrito en el pin digital que es modificado pinchando con un clic sobre el botón es registrado en su correspondiente pin analógico y escrito en las LineEdit (A0..A15).

En la figura 7.9 se han marcado positivamente todas las salidas digitales obteniendo sus valores de lectura en las diferentes entradas analógicas que en este caso son todas 5V. En la figura 7.10 todas a 0V y en la figura 7.11 valores a 5 y a 0V.

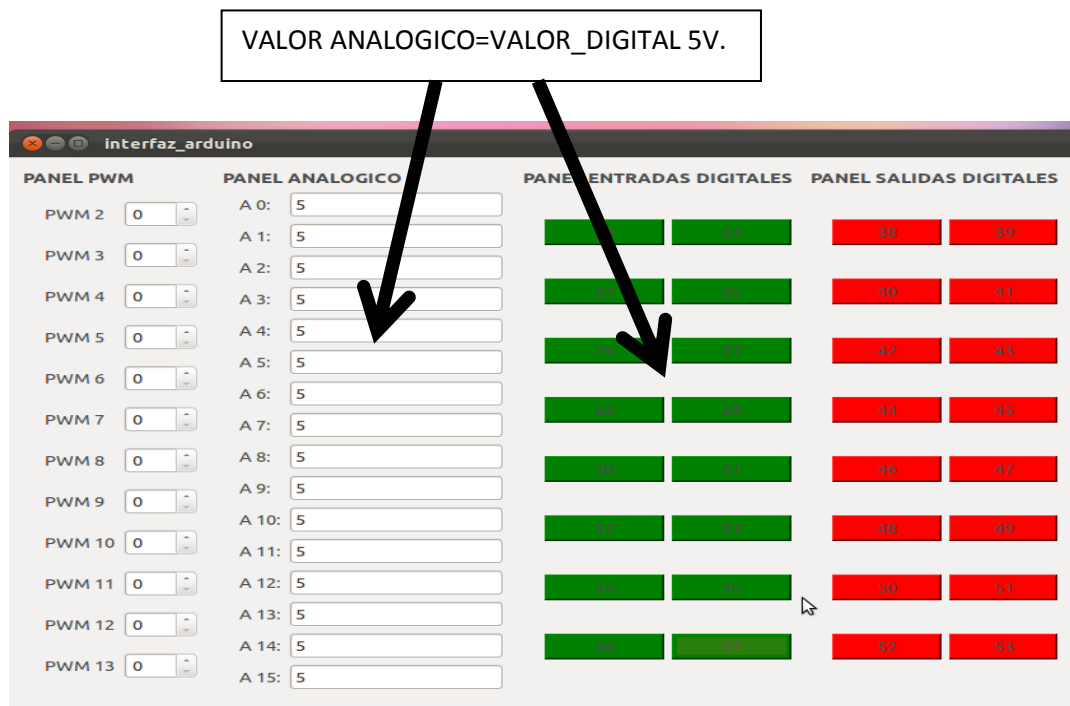


Figura 7. 9. Interfaz salidas a analógicas a 5V

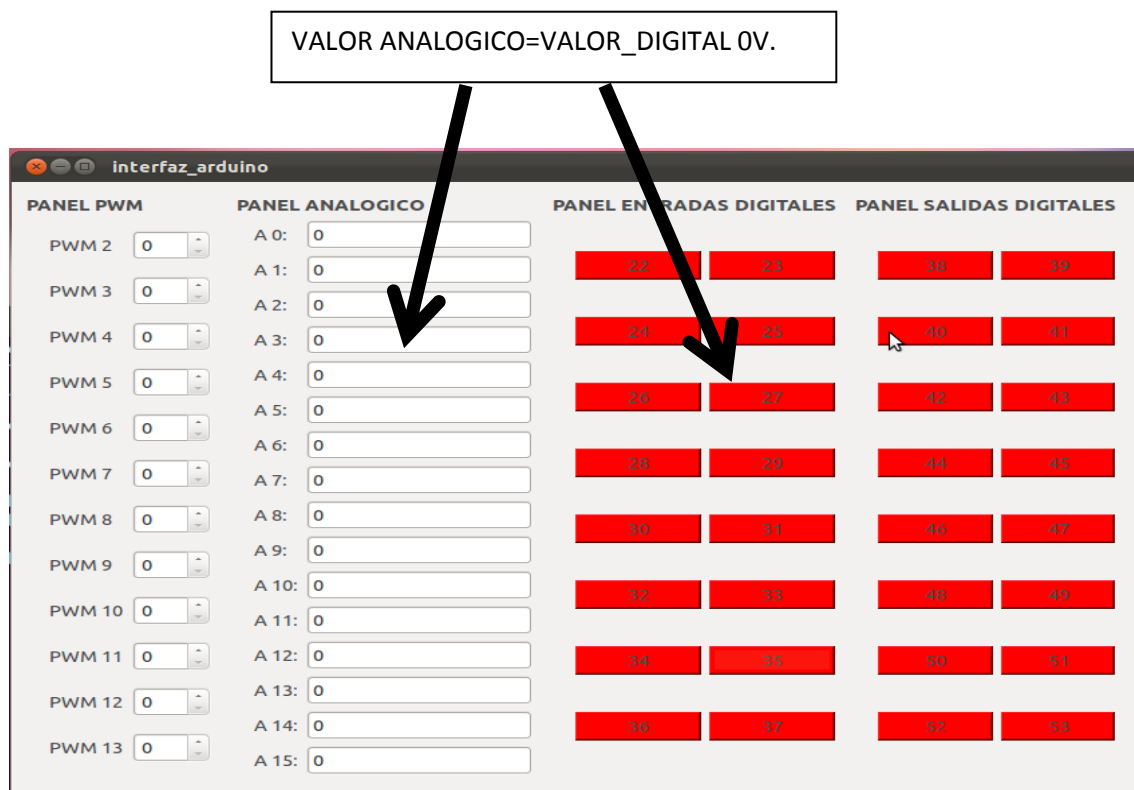


Figura 7. 10. Interfaz salidas a analógicas a 0V

VALOR ANALOGICO=VALOR_DIGITAL 0V o 5v según rojo o verde

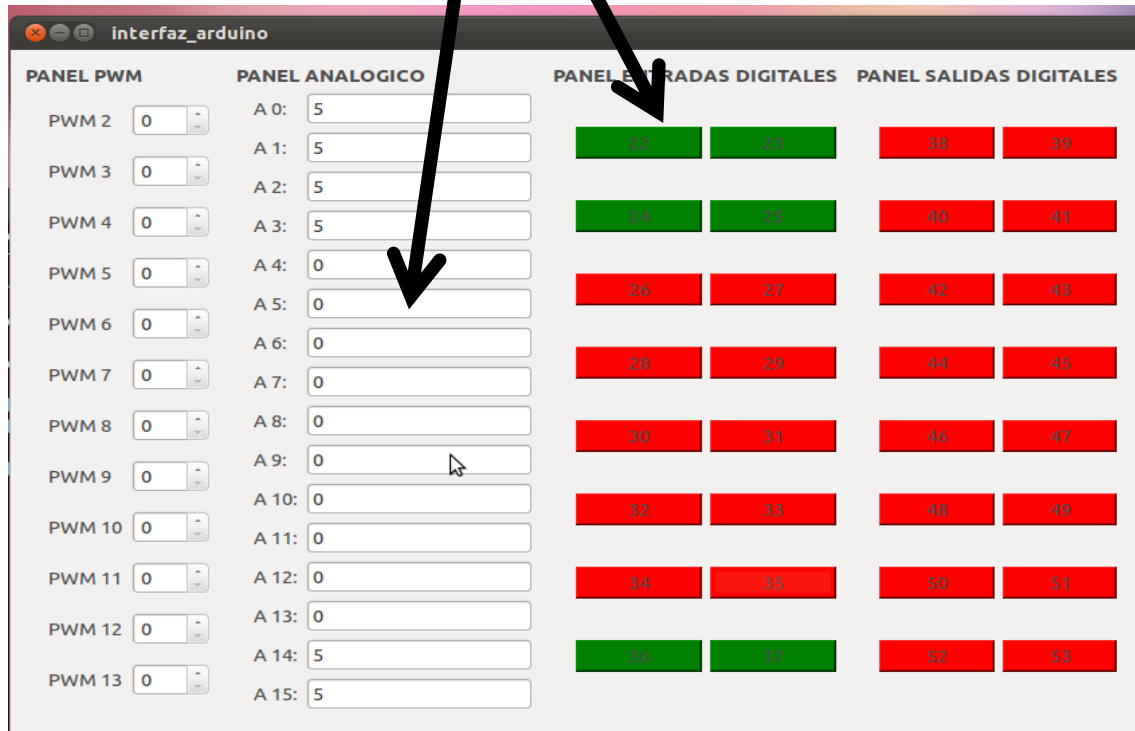


Figura 7. 11. Interfaz salidas a analógicas a 0-5V

Utilizando la herramienta rxplot de ROS obtenemos un osciloscopio básico. Este comando muestra en un gráfica el valor del topic analizado (adquiere el valor del pin digital) frente al tiempo. Utilizamos el comando `rxplot lectura_analogica/ADC0` para mostrar las lecturas registradas en A0 recordemos conectada al pin 22 digital. Hemos producido fluctuaciones de 0 a 5 V para comprobar el funcionamiento (Ver Figura 7.12.).

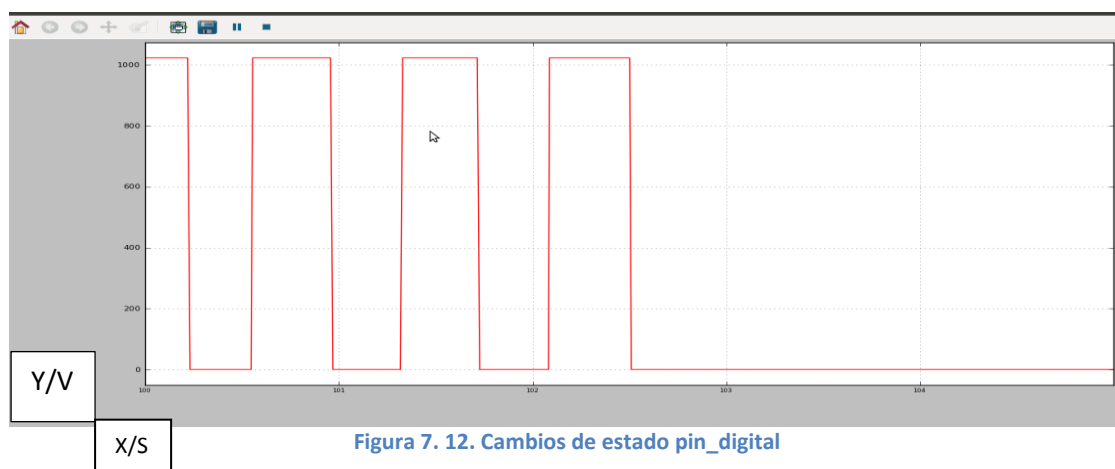


Figura 7. 12. Cambios de estado pin_digital

Recordemos que la resolución del Arduino es de 10 bits con un rango de 0 a 5V por tanto un nivel alto lógico o de 5 V corresponderá con 1023 y un estado lógico bajo o de 0V con un 0.

Entonces en el eje Y el máximo valor alcanzado será de 1023 cuando el pin este a 5V y 0 cuando esté a 0V. En el eje X el tiempo aparece en segundos.

Como se puede apreciar en la figura se han producido unas 3 fluctuaciones por segundo empezando por un valor de 5V y acabando en 0V.

Para comprobar que es capaz de publicar los eventos producidos en la interfaz y a su vez publicarlos en la señal analógica se han producido fluctuaciones más rápidas con un mayor número de clics sobre el botón por segundo (unas 6) como se muestra en la figura 7.13

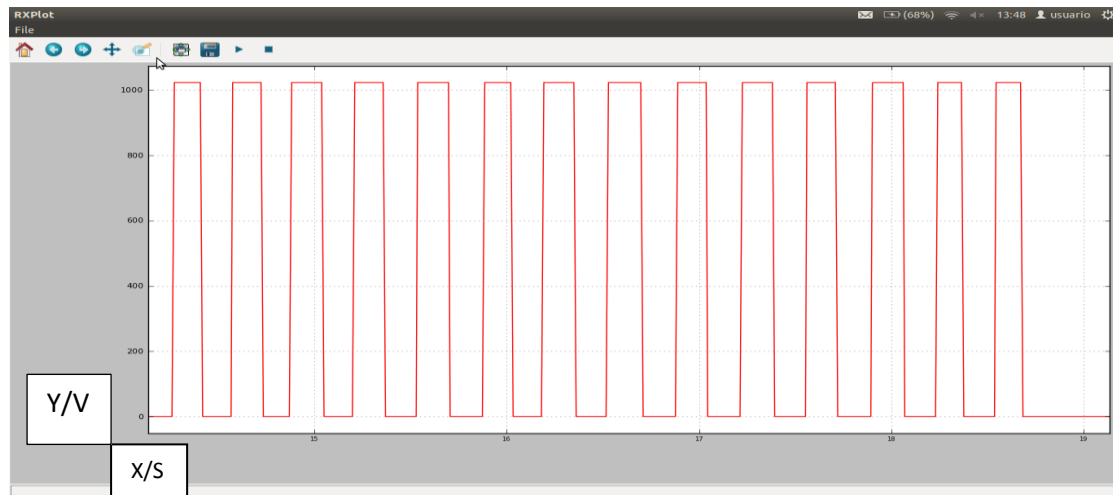


Figura 7. 13. Cambios de estado más veloces

7.5. Entradas analógicas

Para las entradas analógicas se ha utilizado un sensor de proximidad gp2d12 en concreto el modelo 2D120X F 05 alimentado a 5V con un rango de 0 a Vcc como salida y un sensibilidad de 35 m V.

Para el montaje se ha utilizado un filtro paso bajo para intentar atenuar el ruido. El sensor es conectado a alguno de los pines de 5V Y GND del Arduino (cables verde y marrón) y la salida (cable amarillo) es conectado a la entrada del filtro. La entrada analógica se conecta a la salida del filtro.

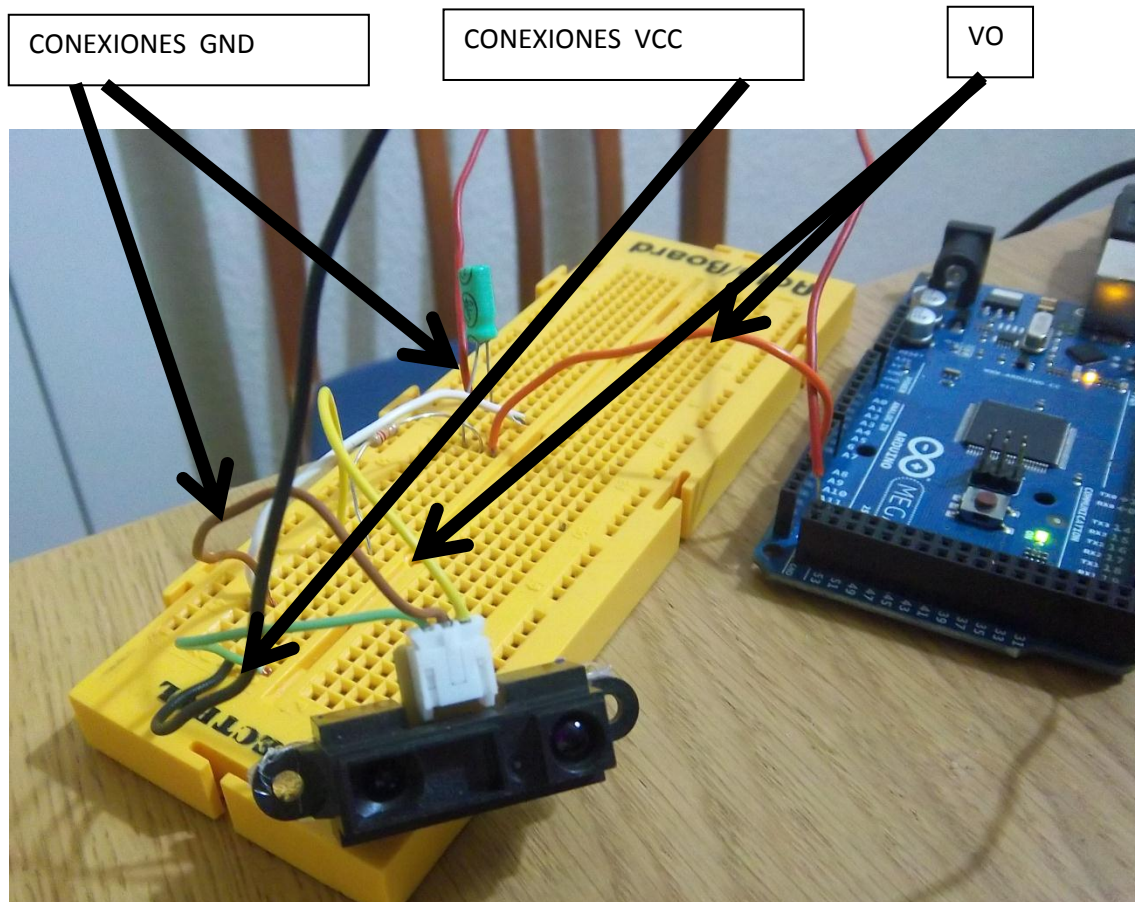


Figura 7. 14. Montaje sensor

Se han examinado 3 casos del sensor a corta, media y larga distancia y se han visualizado los resultados mediante la herramienta rxplot que dibuja los mensajes publicados que adquieren en el valor de la entrada analógica seleccionada. En todas las gráficas el tiempo de las abscisas corresponde con el tiempo en segundos y el eje de las ordenadas con el voltaje con una resolución de 8 bit (0-256)

7.5.1. Distancia larga

Se coloca el sensor a distancia larga, el sensor tiene un rango de actuación de hasta 80 cm por lo tanto colocamos un objeto a aproximadamente esa distancia obteniendo la siguiente medida. Se utiliza la herramienta rxplot para observar los resultados. Como se observa en la figura 7.15

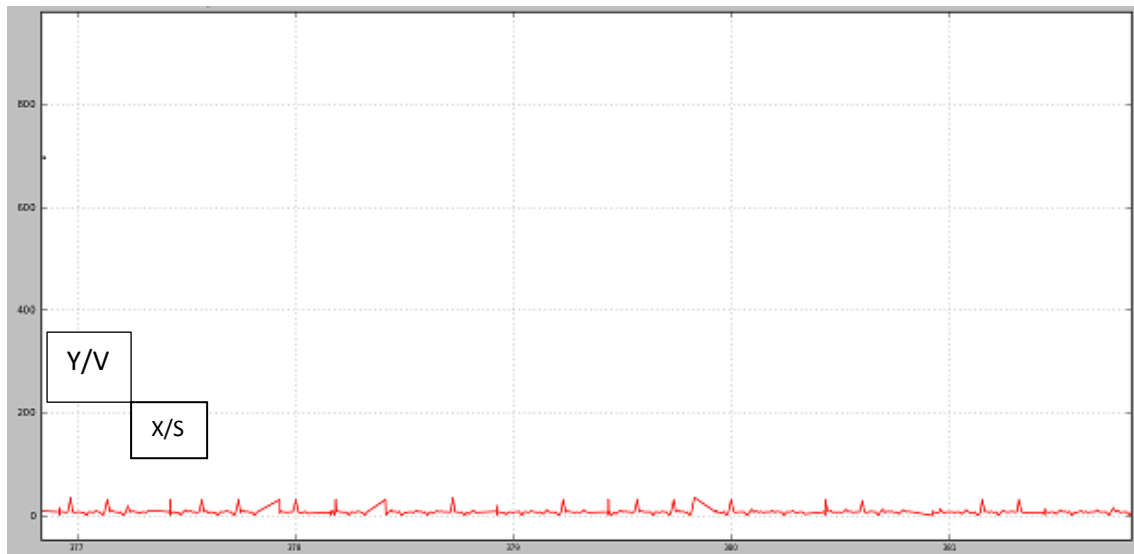


Figura 7.15. Sensor distancia larga

El voltaje es prácticamente 0 ya que no hay ningún objeto que detectar lo suficientemente cerca. Se detecta un ruido que podría ser considerable.

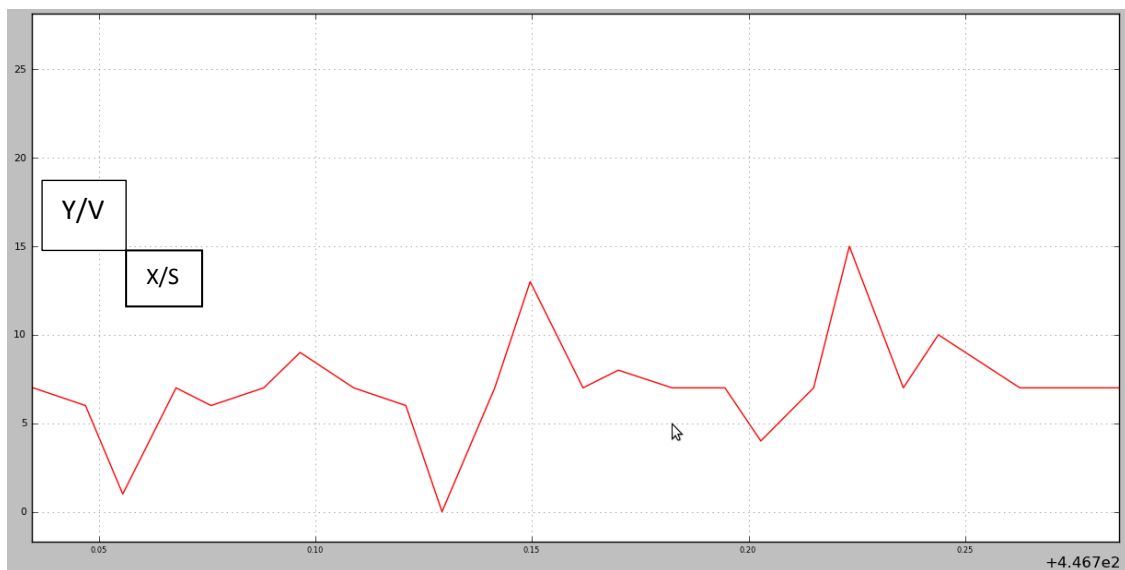


Figura 7.16. Detalle distancia larga

Las diferencias de voltaje entre varias medidas son de 5 unidades, teniendo en cuenta la resolución $5/1023$ obtenemos un ruido de unos 0.025 V. (ver figura 7.16)

7.5.2. Distancia media

En este caso se coloca un objeto delante del sensor a una distancia media generando una tensión proporcional a dicha distancia (figura 7.17).

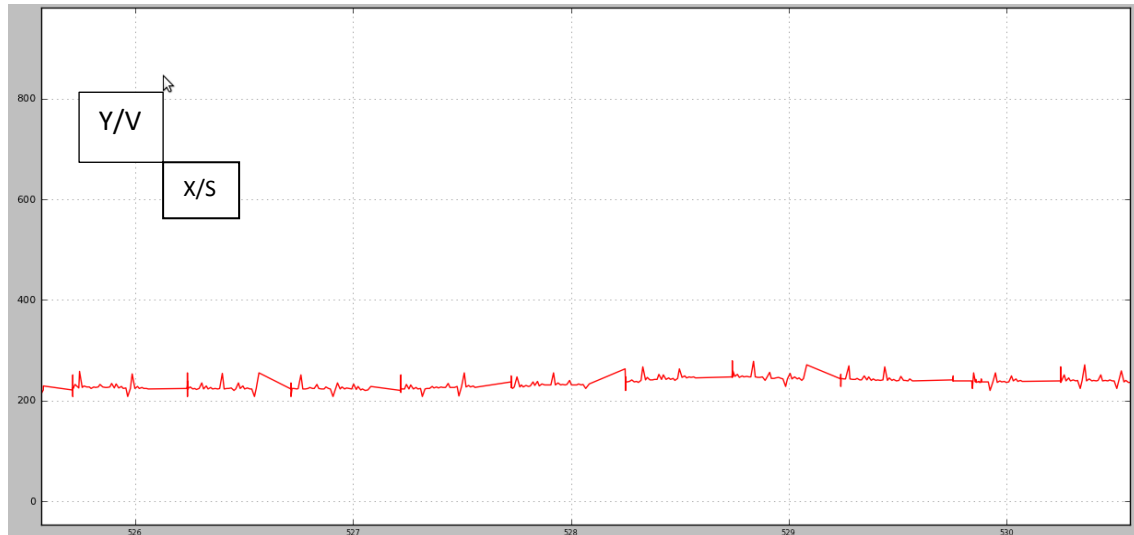


Figura 7.17. Sensor distancia media

Se coloca un objeto a distancia media por lo que recogemos un voltaje de $220 \pm 49 \text{ mV}$ (sensibilidad de la entrada analógica) lo que origina un voltaje de unos 1,08 V. Se muestra un detalle de la grafica en la figura 7.18.

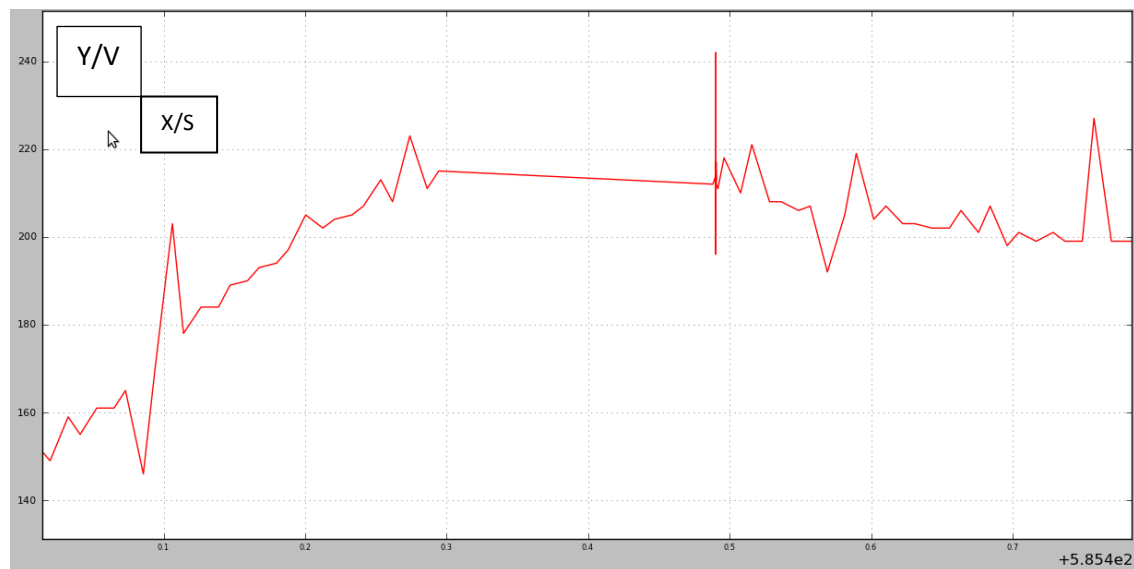


Figura 7.18. Detalle distancia media

7.5.3. Distancia corta

En este caso se coloca un objeto delante del sensor a una distancia corta generando una tensión proporcional a dicha distancia (figura 7.19)

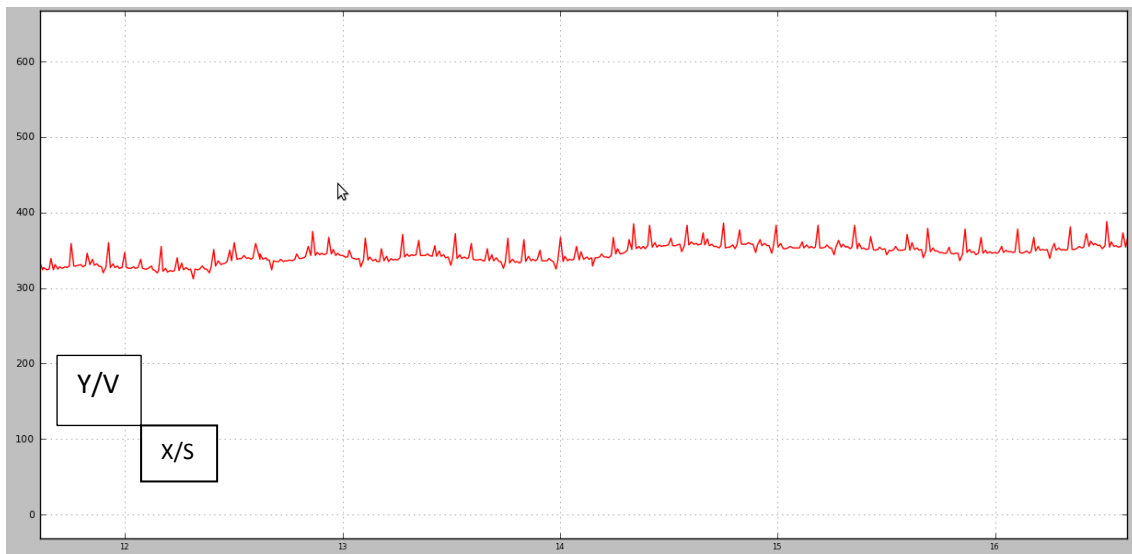


Figura 7. 19. Sensor distancia corta

Se ha aumentado el voltaje con una distancia más corta que las anteriores. Produciéndose una tensión de 1,81 V. Se muestra un detalle de la grafica en la figura 7.20.

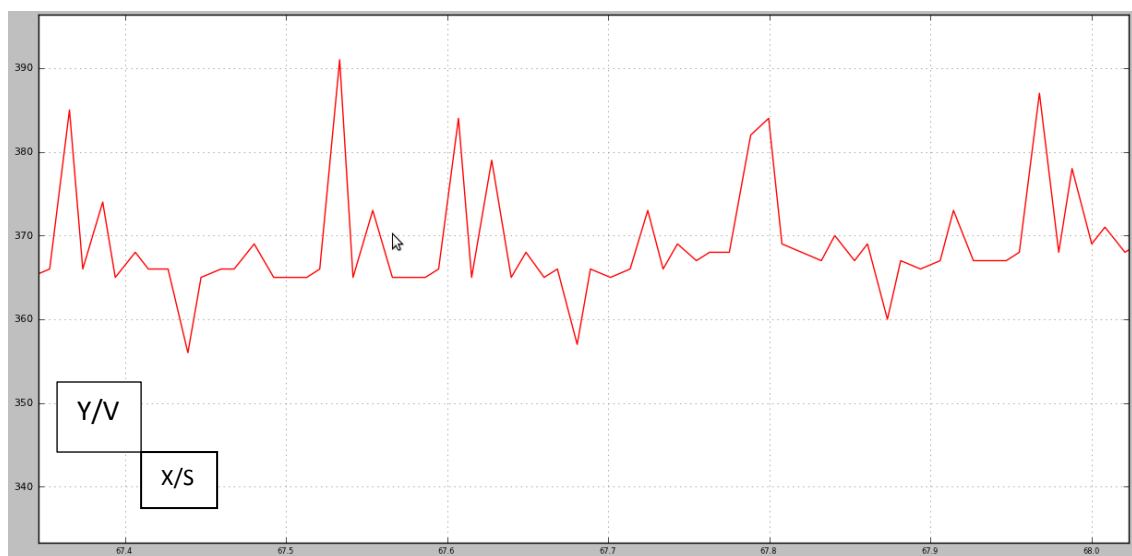


Figura 7. 20. Detalle distancia corta

7.5.4. Fluctuaciones.

Se ha acercado el objeto y luego alejado produciendo fluctuaciones en la medición del sensor. Comprobando que se recogen los datos en un tiempo aceptable (tiempo de publicación). Los resultados obtenidos se muestran en las gráficas siguientes (figuras 7.21, 7.22).

Se lleva el objeto desde una distancia media a una distancia corta para luego volver a alejarlo (figura 7.21).

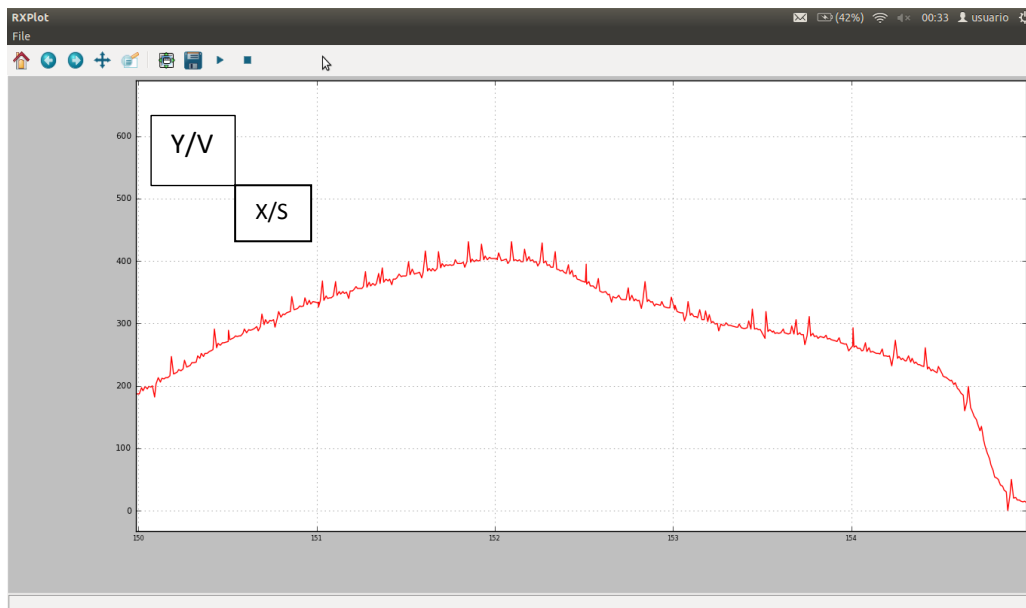


Figura 7. 21. Fluctuación media corta

En la figura 7.22 se aleja y se acerca el objeto mas rápidamente obteniendo una variación de tensión más rápida.

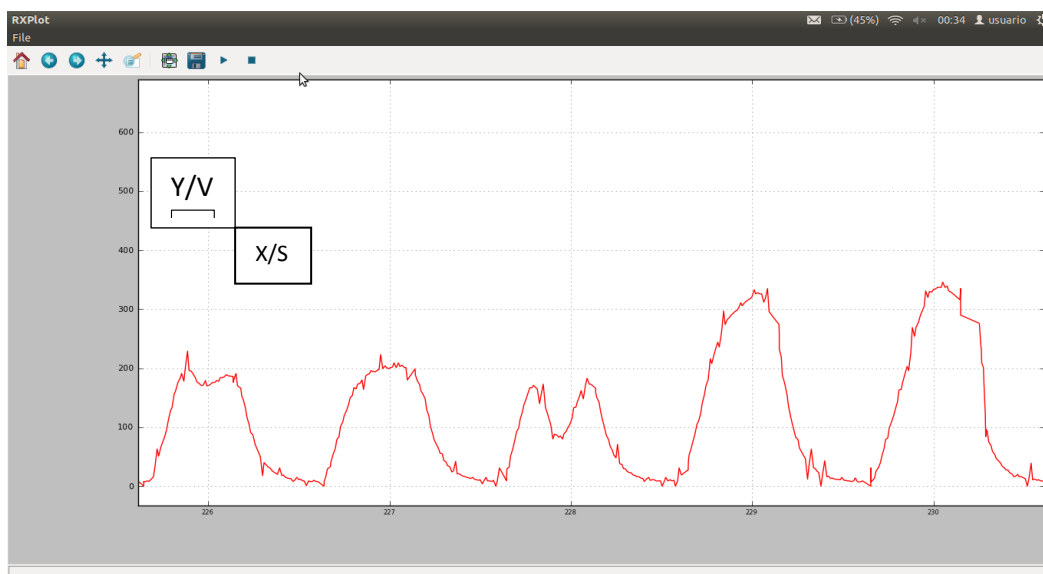


Figura 7. 22. Fluctuaciones rápidas

7.6. Salidas PWM

PWM (Pulse Width Modulated). Es una modulación de ancho del pulso, es una señal de onda cuadrada con una frecuencia constante y una duración del pulso variable. Dependiendo de la relación entre el ancho del pulso y el intervalo del mismo (tiempo de duración del pulso sobre periodo del mismo), un ancho de pulso corto produce una baja corriente efectiva, y un ancho de pulso largo produce una alta corriente efectiva. Se utiliza para la alimentación de aparatos eléctricos como motores o lámparas.

Arduino proporciona una frecuencia configurada de aproximadamente 490 Hz, pudiéndose cambiar este mediante los timers internos del ATMEGA.

Para la medición de las salidas pwm primero se ha obtenido por pantalla la salida que ofrece esta conectándola con un entrada analógica y luego visualizando los datos mediante rxplot. Se ha variado únicamente el ciclo de trabajo en la interfaz de usuario.

7.6.1. Casos analizados

Para la prueba se ha realizado 5 casos variando el ciclo de trabajo. La frecuencia se mantiene constante a aproximadamente 490 Hz:

CICLO DE TRABAJO	10%	25%	50%	75%	95%
FRECUENCIA	490 HZ	490HZ	490 HZ	490 HZ	490 HZ

- Primer caso: 10%

Con este ciclo de trabajo el tiempo de pulso se mantiene a nivel alto durante una décima parte. Como se observa en la figura (7.23). La frecuencia no es del todo exacta debido a que se trata de publicaciones de ros y éste publica con un ratio determinado.

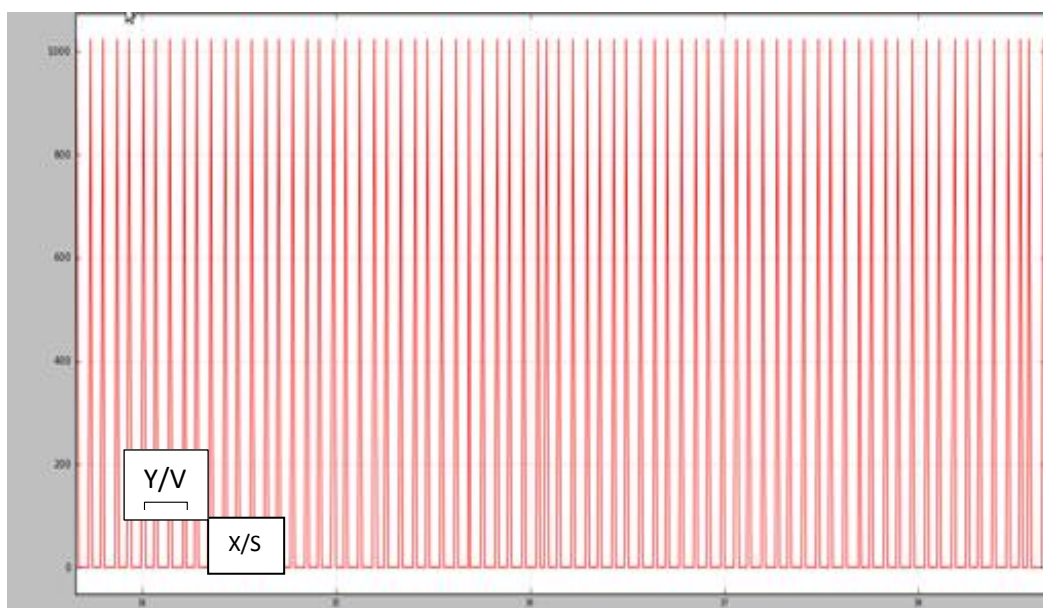


Figura 7. 23. Ciclo de trabajo al 10%

- Segundo caso: 25%

Se observa que los pulsos a nivel alto son de mayor duración que los de la figura anterior. Correspondería aproximadamente con un cuarto a nivel alto y el resto del tiempo a nivel bajo. Ver figura 7.24.

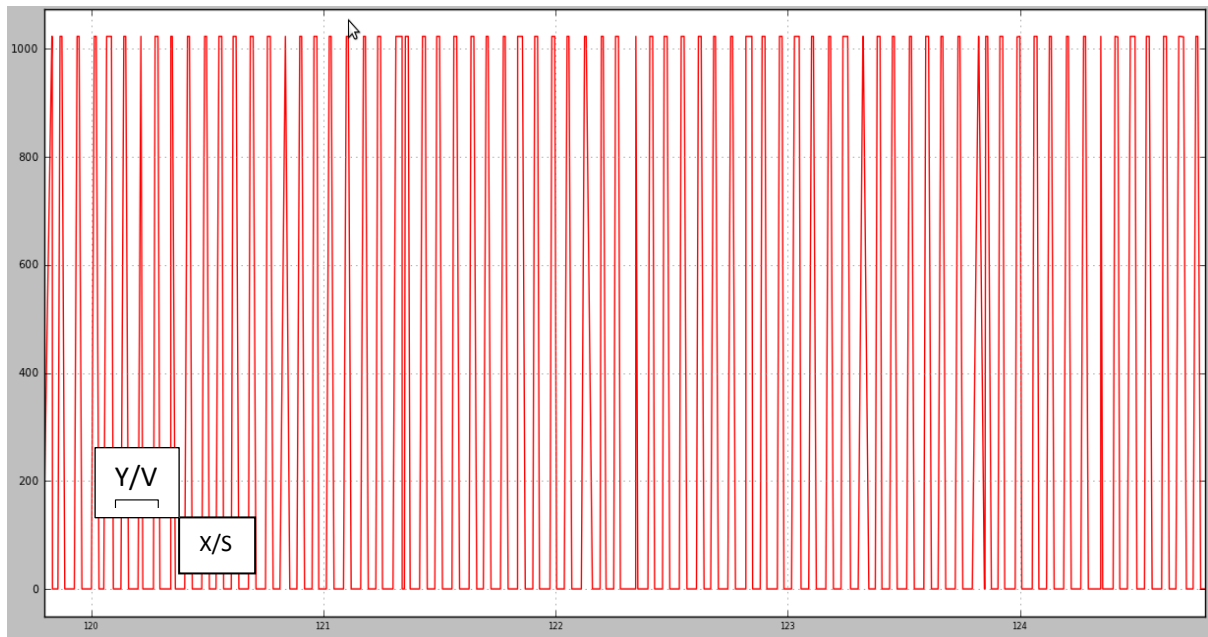


Figura 7. 24. Ciclo de trabajo al 25%

Se repite el proceso para un ciclo de trabajo del 50% en la figura 7.25

- Tercer caso: 50%

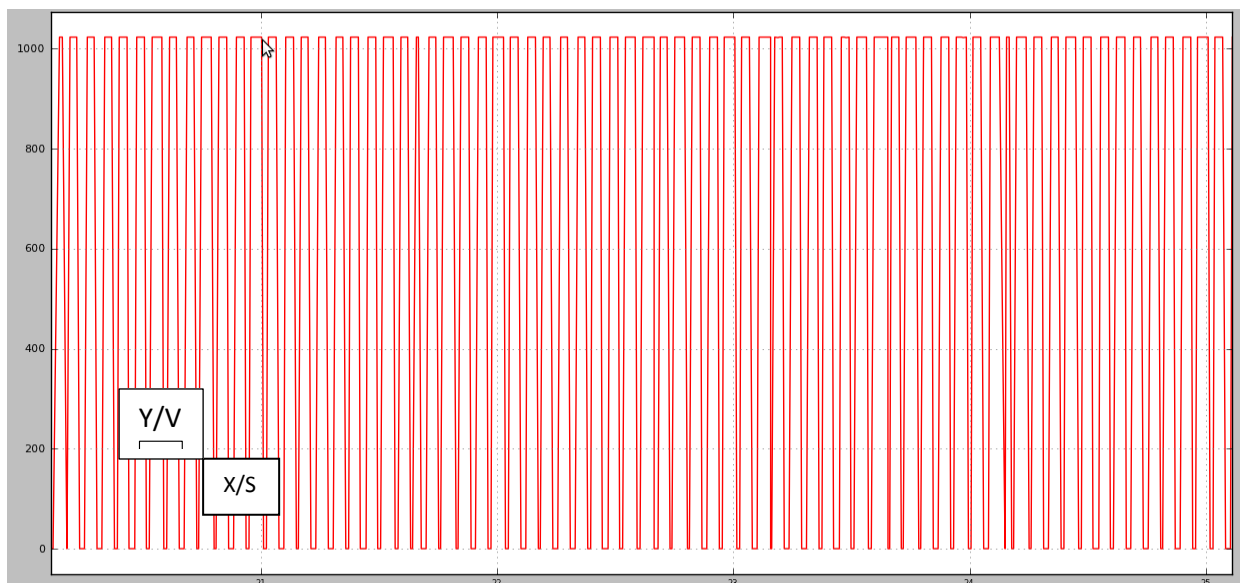


Figura 7. 25. Ciclo de trabajo al 50%

Con el ciclo de trabajo al 50% se obtiene una PWM que corresponde con un detalle de la figura 7.25.

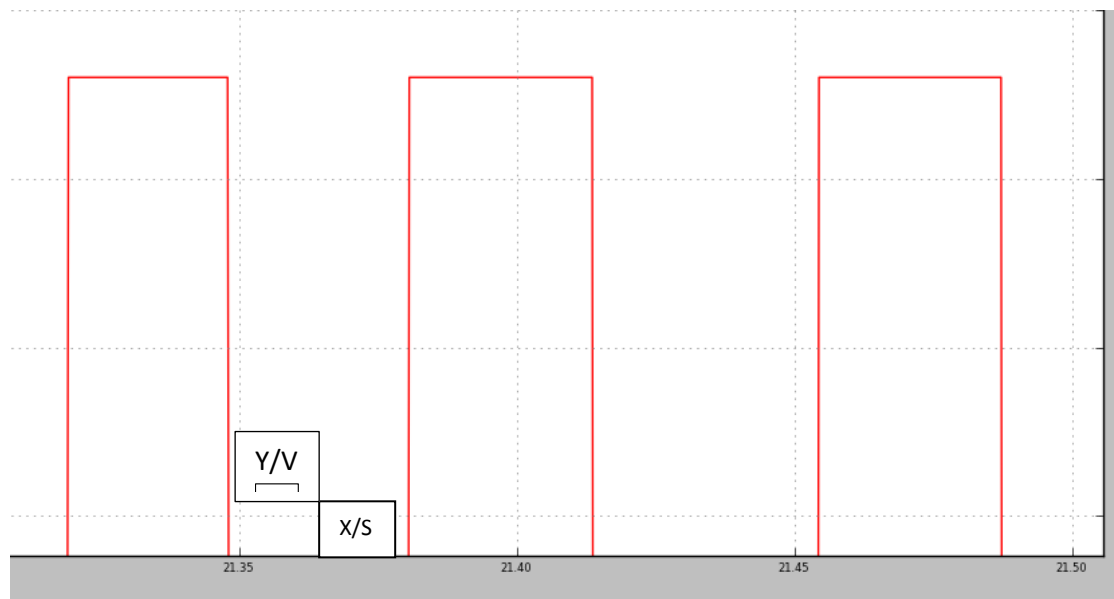


Figura 7. 26. Detalle del caso al 50%

Se repite el proceso para un ciclo de trabajo del 75% en la figura 7.27

- Cuarto caso: 75%

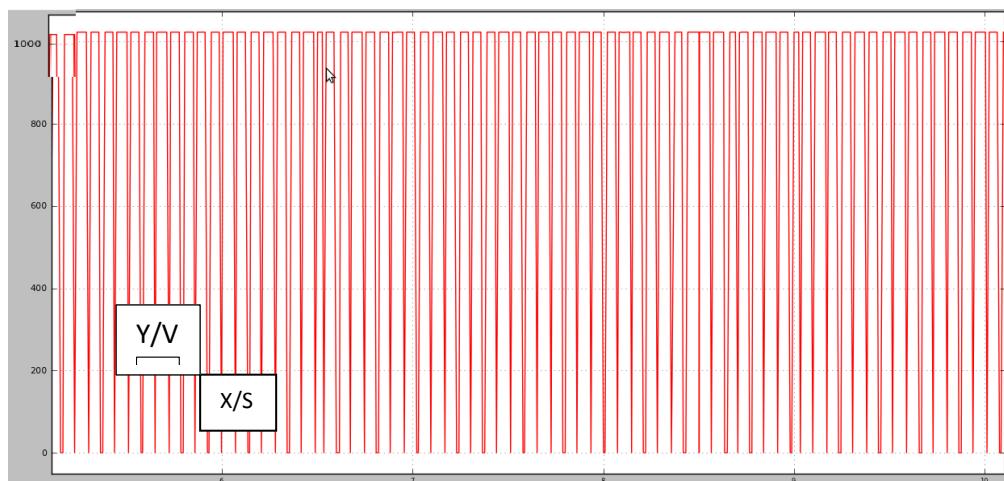


Figura 7. 27. Ciclo de trabajo al 75%

En este caso se corresponde con $\frac{3}{4}$ partes del tiempo a 5 V y $\frac{1}{4}$ a 0 V. Ver figura 7.28.

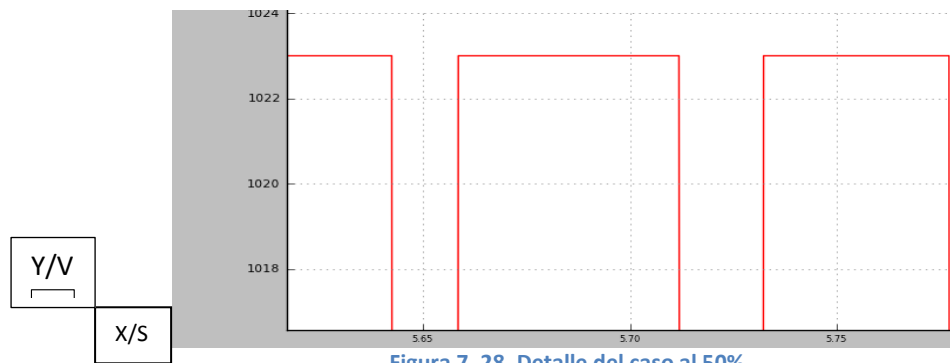


Figura 7. 28. Detalle del caso al 50%

- Quinto caso: 95%

Se repite el proceso para un ciclo de trabajo del 95% en la figura 7.29. Se observan que algunas mediciones de 0 V no son obtenidas en ROS por el retardo de publicación de los mensajes como se muestra en la figura 7.29.

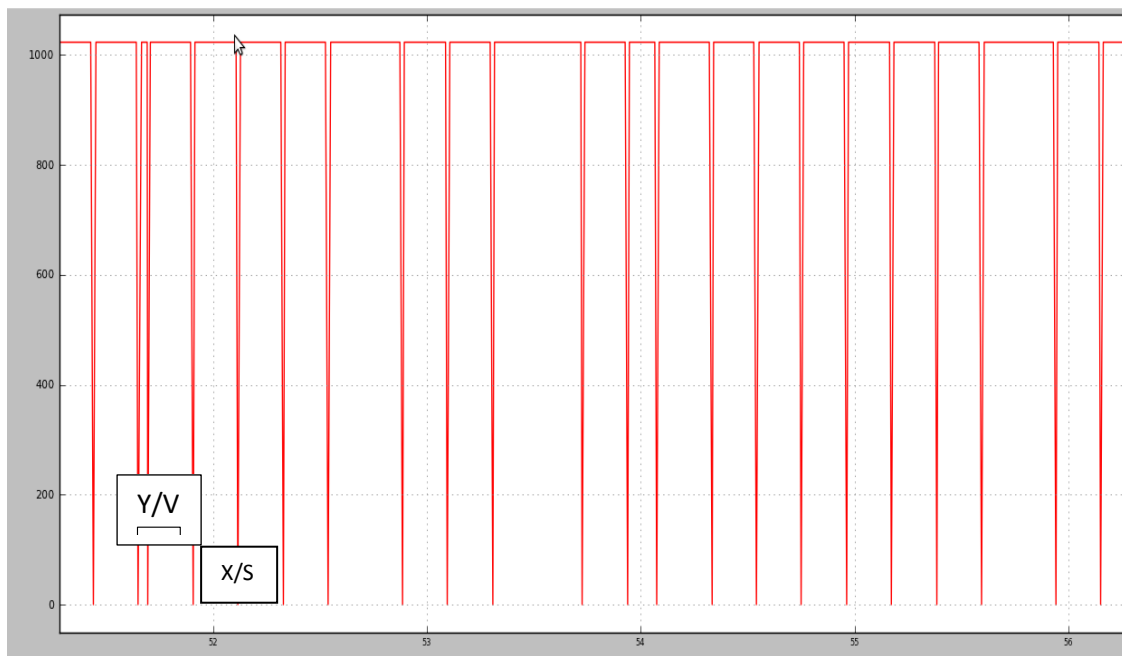


Figura 7. 29. Ciclo de trabajo al 95%

Se muestra el detalle de la PWM con ciclo de trabajo al 95% en el que el tiempo de valor 0 corresponde con un 5%.

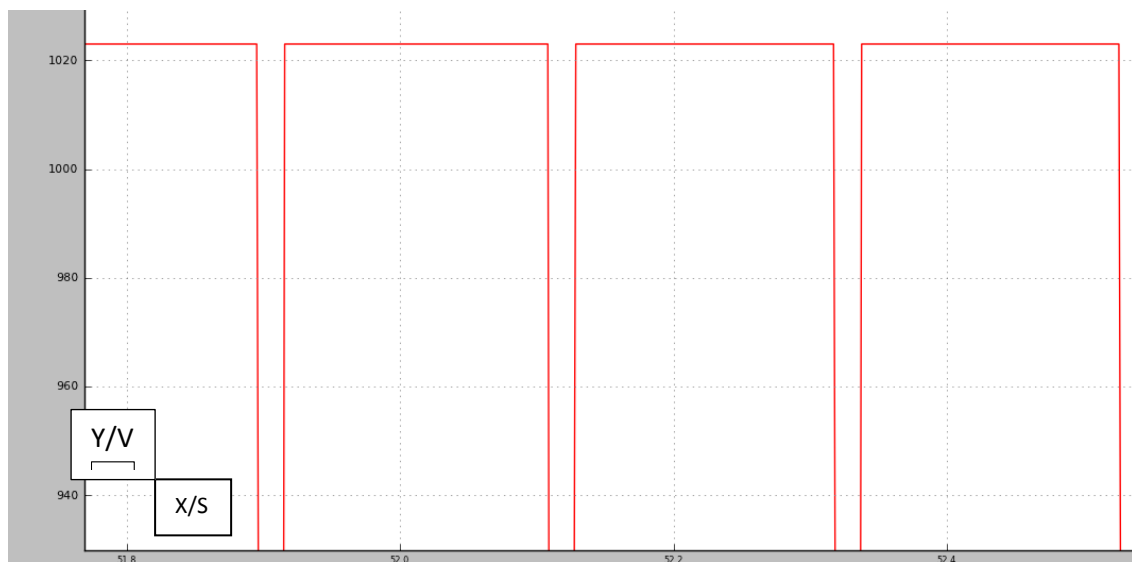


Figura 7. 30. Detalle del caso al 95%

7.7. Mediciones varias PWM

Se han conectado las salidas PWM a varios LED controlando su ciclo de trabajo estos se iluminaran más o menos debido a la diferentes intensidades recibidas y controladas por el ciclo de trabajo de las PWM. De derecha a izquierda se han aumentado los ciclos de trabajo a los que están conectados los LED luciendo estos con mayor intensidad como muestra la figura 7.31

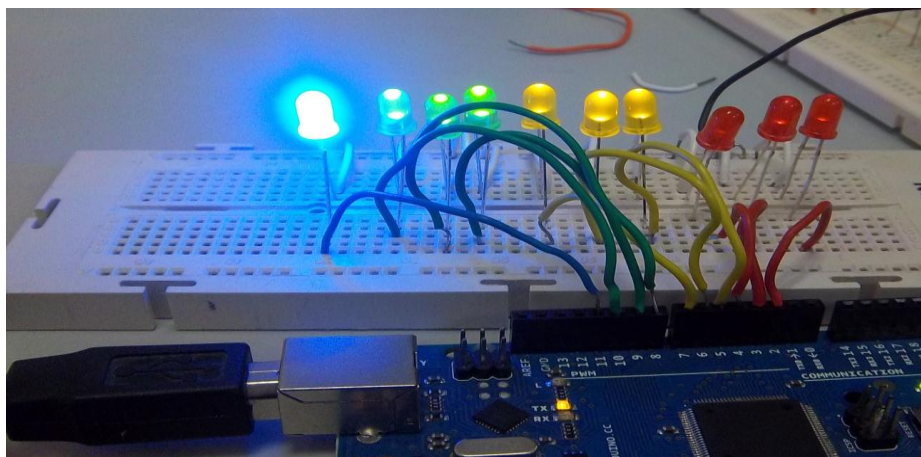


Figura 7. 31. Múltiples LED's

Se muestran los valores escritos en la interfaz Arduino en el "PANEL PWM" desde 5 a 99 como se muestra en la figura 7.32

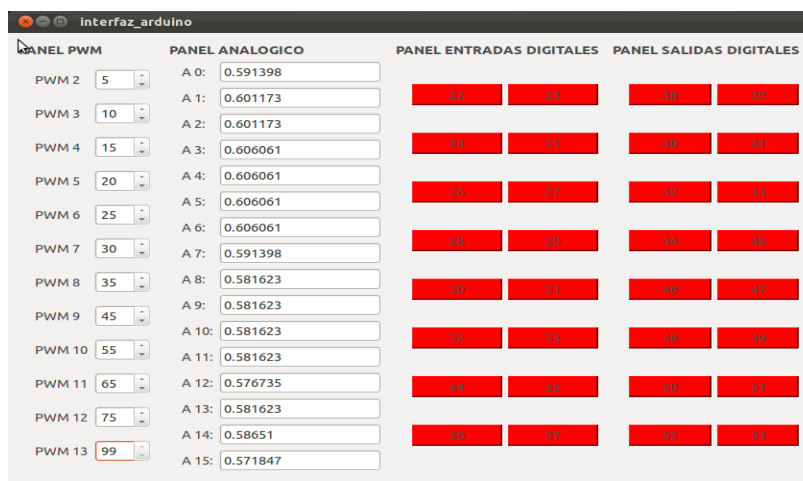


Figura 7. 32. Interfaz para controlar múltiples LED's

Con todas estas pruebas se ha comprobado el correcto funcionamiento de la placa Arduino integrada con ROS. De esta manera al realizar cambios en la interfaz la placa actualizara sus salidas y a su vez enviara la información de sus entradas, todo ello mediante comunicacion ROS.



Capítulo 8: CONCLUSIONES Y TRABAJOS FUTUROS.

8.1. Conclusiones.

Se ha conseguido la integración de Arduino con ROS creando un nodo capaz de publicar mensaje continuamente, actualizando las entradas de Arduino, cuyos mensajes pueden ser recibidos por cualquier otro nodo ROS lo que implica una compatibilidad para la integración de esta placa en sistemas robóticos. Dentro de este nodo se han logrado también la subscripción a mensajes que permiten la modificación de sus salidas. Esta modificación de salidas se produce mediante interrupciones

Por otro lado se ha conseguido la implementación de una interfaz de usuario que visualiza las entradas de la placa y permite escribir de una forma sencilla y visual valores para las salidas de la misma. Esta interfaz de usuario representa un ejemplo de nodo que permite la comunicación directa con ROS y Arduino pero que además ofrece una forma sencilla de manipulación y visualización de los datos.

En cuanto a la adquisición de datos se ha comprado las características de la placa en cuanto a resolución, frecuencia de muestreo y calidad. En lo concerniente a la resolución Arduino presenta una ventaja frente a las placas ya existentes y es que permite cambiar fácilmente la tensión de referencia de sus pines. La frecuencia de muestreo se ha analizado a la frecuencia determinada en Arduino de 490 Hz y aunque éste puede ser aumentado considerablemente equiparándola a un DAQ, escribiendo en la memoria interna del microcontrolador, exprimiendo así las posibilidades del microcontrolador esto sólo serviría para lecturas de tiempo cortas debido a las características físicas de la memoria. La calidad por otro lado es lo suficientemente buena en los casos analizados de esta frecuencia determinada.

Por todo ello se ha conseguido una interfaz de usuario que opera sobre el Arduino y pensada para trabajar en sistemas acoplados en Robots.

8.2. Trabajos futuros

- Realizar una configuración dinámica de las entradas y salidas de la placa incluyendo en la interfaz de usuario un panel de configuración dinámico que permita la modificación de estas según las especificaciones requeridas.
- Conseguir modificar la referencia de los pines y la frecuencia a la que son generadas determinadas salidas actuando sobre los timers internos de la placa.
- Conseguir una mejor respuesta del sistema ROS-ARDUINO en cuanto a tiempos de comunicación y velocidad de lectura actuando sobre la memoria interna del microcontrolador
- Explorar las diferentes herramientas que proporciona ROS para mejorar lecturas y visualizar datos.
- Permitir utilizar la memoria interna de Arduino y ROS para almacenar datos de relevancia.
- Utilizar los demás pines de la placa en especial los puertos series para conseguir la integración con otros sistemas, consiguiendo un sistema de comunicación múltiple.



Referencias

- [1] <http://arduino.cc/es/Main/Software> desde aquí se puede descargar el software para empezar a utilizar Arduino. Visualizada noviembre 2012
- [2] <http://arduino.cc/es/> , página web de Arduino en español. No todos los apartados que tiene en inglés están traducidos. Consultada por última vez el día. Visualizada octubre 2012
- [3] <http://arduino.cc/forum/index.php?board=32.0> , foro de la página de Arduino en español. Consultada por última vez el día. Visualizada noviembre 2012
- [4] <http://arduino.cc> , página web de Arduino en inglés. Está más actualizada. Visualizada noviembre 2012
- [5] <http://arduino.cc/en/Hacking/Programmer>. Visualizada noviembre 2012
- [6] Libro “Arduino programming notebook”. Autor Brian W. Evans. Editorial “Creative Commons”. Segunda edición, 2008. Visualizada noviembre 2012. Visualizada noviembre 2012
- [7] <http://arduino.cc/es/Tutorial/HomePage>, librerías compatibles con Arduino. Consultada por última vez el día. Visualizada noviembre 2012. Visualizada septiembre 2012
- [8] <http://qt-project.org/> Comunidad Qt Project. Visualizada noviembre 2012
- [9] <http://qt.digia.com/product/> , página web de QT. Visualizada noviembre 2012
- [10] <http://qt.digia.com/Try-Qt-Now/> , desde aquí se puede descargar el programa QT. Visualizada noviembre 2012
- [11] <http://info.qt.digia.com/contact-qt/> preguntas o sugerencias acerca de QT. Visualizada noviembre 2012
- [12] <http://qt.digia.com/product/>. Visualizada noviembre 2012
- [13] <http://www.ros.org/wiki/> .página web de ROS. Visualizada noviembre 2012
- [14] <http://www.ros.org/wiki/ROS/Installation> desde aquí se puede descargar el software para empezar a utilizar ROS. Visualizada septiembre 2012
- [15] <http://www.ros.org/wiki/APIs> bibliotecas cliente de ROS. Visualizada septiembre 2012
- [16] http://www.ros.org/wiki/rosserial_arduino paquete roserial Arduino. Visualizada septiembre 2012
- [17] <http://www.ros.org/wiki/Support?action=show&redirect=Mailing+Lists>. Visualizada noviembre 2012
- [18] <http://answers.ros.org/questions/>. Visualizada noviembre 2012

ANEXOS.

ANEXO 1. Ros/qt

A1.1.MAIN.CPP

```
#include <QApplication>
#include "ros/ros.h"
#include "arduino.h"
#include <QLabel>
#include "nodo.h"
#include <QThread>
int main(int argc, char *argv[])
{

    QApplication app(argc, argv);
    Arduino dat(argc,argv);
    Nodo nodo(argc, argv, "arduino");

    dat.show();
```

A.1.2. ARDUINO.H

```
#ifndef ARDUINO_H
#define ARDUINO_H

#include <QWidget>
#include <QPushButton>
#include <QSpinBox>
#include <QSlider>
#include <QLabel>
#include <QDialog>
#include "nodo.h"
#include <std_msgs/String.h>
#include <std_msgs/Int16.h>
#include <string>
#include <std_msgs/Int8.h>

class Pin;
class QGroupBox;

class Arduino : public QWidget
{
    Q_OBJECT
public:

    Arduino(int argc, char *argv[],QWidget *parent = 0);
    int e_digital();
    int calcular();

signals:
    void digitClicked(int digit);
    void valueChanged(int value);
public slots:

void modificar_analogico(float v0,float v1,float v2,float
v3,float v4,float v5,float v6,float v7,float v8,float v9,float
v10,float v11,float v12,float v13,float v14,float v15);
void digitClicked1(int digit);
void escribir_pwm(int value,int numero);
    void paintEvent(QPaintEvent *);
void chatterCallback(std_msgs::String);
void modificar_pin(int valor_entradas);

private:

void crear_analogico();
void crear_digital();
```

```
void crear_pwm();
void crear_imagen();

int referencia_analog;
bool pines[31];
QLabel*ana[16];
QLabel*label[12];
QSlider * slider[11];
    QSpinBox *pwm[11];
QPushButton*buttons[31];
QLineEdit *lineEdit[16];
int mydigit;
int value;
int init_argc;
char** init_argv;

int valor_pwm[11];
bool valor[31];

    QGroupBox*imagen;
    QGroupBox*digital;
    QGroupBox*digital2;
    QGroupBox *analogico;
    QGroupBox*PWM;

std_msgs::Int32 pin_msg;
ros::Publisher chatter_publisher;
ros::Publisher chatter_publisher2;
ros::Subscriber sub;
ros::Subscriber sub2;
int dato;
Nodo nodo;

};

#endif
```


A1.3. ARDUINO.CPP

```
#include <QGridLayout>
#include <std_msgs/Int16.h>
#include <ros/ros.h>
#include <QtGui>
#include "arduino.h"
#include "pin.h"
#include "pwm.h"
#include <QLabel>
#include "nodo.h"
#include "interfaz_arduino/pwm.h"
#include <std_msgs/String.h>
#include <string>
#include <std_msgs/Int8.h>

Arduino::Arduino(int argc, char *argv[],QWidget *parent) :
QWidget(parent)
,nodo(argc,argv)
{

    crear_pwm();
    crear_analogico();
    crear_imagen();

referencia_analog=0;

for (int i = 0; i < 16; ++i) {
buttons[i] = new Pin(i,this);
connect(buttons[i], SIGNAL(clicked(int)), this,
SLOT(digitClicked1(int)));

}

digital = new QGroupBox(tr("PANEL ENTRADAS DIGITALES"));
QGridLayout *layout = new QGridLayout;
for (int i = 0; i < 16; i=i+2) {
layout->addWidget(buttons[i], i + 1, 0);
layout->addWidget(buttons[i+1], i + 1, 1);

}
digital->setLayout(layout);

for (int i = 16; i < 32; i++) {
buttons[i] = new Pin(i,this);
}

digital2 = new QGroupBox(tr("PANEL SALIDAS DIGITALES"));
```

```
QGridLayout *layout2 = new QGridLayout;

for (int i = 16; i < 32; i=i+2) {
layout2->addWidget(buttons[i], i + 2, 0);
layout2->addWidget(buttons[i+1], i + 2, 1);
}

digital2->setLayout(layout2);

//inicializacion pin
for(int i=0; i<16;i++){
buttons[i]->setStyleSheet("background : red");
valor[i]=0;
}
for(int i=0; i<12; i++){
valor_pwm[i]=0;
}

QObject::connect(&nodo, SIGNAL(cambiar_pin(int)), this,
SLOT(modificar_pin(int)));
QObject::connect(&nodo,
SIGNAL(cambiar_analogico(float,float,float,float,float,float,float,
float,float,float,float,float,float,float,float)), this,
SLOT(modificar_analogico(float,float,float,float,float,float,float,
float,float,float,float,float,float,float,float)));

ros::start(); // necesario para nodos ros fuera del terminal
ros::NodeHandle n;

// comunicacion ros publicadore.
 chatter_publisher =
n.advertise<std_msgs::Int16>("entradas_digitales", 1000);
 chatter_publisher2 =
n.advertise<interfaz_arduino::pwm>("pwm", 1000);

QGridLayout *mainLayout = new QGridLayout;

mainLayout->addWidget(PWM,0,0);
mainLayout->addWidget(analogico,0,1);
mainLayout->addWidget (digital,0,2);
mainLayout->addWidget(digital2,0,3);
setLayout(mainLayout);

QLabel *ard = new QLabel;
```

```
QImage image(":/images/arduino.jpg"); //indicas donde esta la
imagen
ard->setPixmap(QPixmap::fromImage(image));
ard->setScaledContents(true);

std_msgs::Int16 pin_msg;
interfaz_arduino::pwm pwm_msg;
    chatter_publisher.publish(pin_msg);
    chatter_publisher2.publish(pwm_msg);
    ros::spinOnce();

nodo.iniciar();
}
void Arduino::digitClicked1(int digit)
{

    switch (valor[digit-22]){
    case 0:
        buttons[digit-22]->setStyleSheet("background : green");

        valor[digit-22]=1;
        pin_msg.data=calcular();

    break;
    case 1:
        buttons[digit-22]->setStyleSheet("background : red");

    valor[digit-22]=0;
    pin_msg.data=calcular();
    break;
    }
    chatter_publisher.publish(pin_msg);
    ROS_INFO("%d", pin_msg.data);
}

void Arduino::modificar_pin(int valor_entradas){
int aux=1;
for(int i=16; i<32;i++){
    if((valor_entradas & aux)==aux){

        buttons[i]->setStyleSheet("background : green");
    }
    else{

        buttons[i]->setStyleSheet("background : red");
    }
    aux=aux*2;

}
}
```

```
int Arduino::calcular(){
int dato=0;
int aux=0;
int n=0;
for(int j=0;j<16;j++)
{
    aux=valor[j];
    dato=dato| (aux<<n);

n++;
}
return dato;

}

void Arduino::escribir_pwm(int value,int numero){
interfaz_arduino::pwm pwm_msg;
valor_pwm[numero]=value;
ROS_INFO("%d",numero);
pwm_msg.pwm2=valor_pwm[0];
pwm_msg.pwm3=valor_pwm[1];
pwm_msg.pwm4=valor_pwm[2];
pwm_msg.pwm5=valor_pwm[3];
pwm_msg.pwm6=valor_pwm[4];
pwm_msg.pwm7=valor_pwm[5];
pwm_msg.pwm8=valor_pwm[6];
pwm_msg.pwm9=valor_pwm[7];
pwm_msg.pwm10=valor_pwm[8];
pwm_msg.pwm11=valor_pwm[9];
pwm_msg.pwm12=valor_pwm[10];
pwm_msg.pwm13=valor_pwm[11];
 chatter_publisher2.publish(pwm_msg);
//ROS_INFO("%d", pwm_msg.data);
}

void Arduino::crear_analogico(){

    analogico = new QGroupBox(tr("        PANEL ANALOGICO"));
    QGridLayout *layout = new QGridLayout;

    for (int i = 1; i < 17; i++) {
        ana[i] = new QLabel(tr("A %1:").arg(i-1));
        lineEdit[i] = new QLineEdit;
        layout->addWidget(ana[i], i + 1, 0);
        layout->addWidget(lineEdit[i], i + 1, 1);
    }

    analogico->setLayout(layout);

}

void Arduino::crear_pwm(){
```

```
PWM = new QGroupBox(tr( "PANEL PWM"));

    QGridLayout *layout = new QGridLayout;
    for(int i=0; i<12; i++){
        pwm[i] = new Pwm(i,this);
        connect(pwm[i], SIGNAL(emitir_pwm(int,int)), this,
        SLOT(escribir_pwm(int,int)));
    }

    for (int i=0; i<12; i++){
        label[i] = new QLabel(tr("PWM %1").arg(i + 2));
        layout->addWidget(pwm[i], i + 1, 1);
        layout->addWidget(label[i], i + 1, 0);
    }

    PWM->setLayout(layout);

}

void Arduino::modificar_analogico(float v0,float v1,float v2,float
v3,float v4,float v5,float v6,float v7,float v8,float v9,float
v10,float v11,float v12,float v13,float v14,float v15){

    lineEdit[1]->setText(QString::number(v0));
    lineEdit[2]->setText(QString::number(v1));
    lineEdit[3]->setText(QString::number(v2));
    lineEdit[4]->setText(QString::number(v3));
    lineEdit[5]->setText(QString::number(v4));
    lineEdit[6]->setText(QString::number(v5));
    lineEdit[7]->setText(QString::number(v6));
    lineEdit[8]->setText(QString::number(v7));
    lineEdit[9]->setText(QString::number(v8));
    lineEdit[10]->setText(QString::number(v9));
    lineEdit[11]->setText(QString::number(v10));
    lineEdit[12]->setText(QString::number(v11));
    lineEdit[13]->setText(QString::number(v12));
    lineEdit[14]->setText(QString::number(v13));
    lineEdit[15]->setText(QString::number(v14));
    lineEdit[16]->setText(QString::number(v15));

}
```

A.1.4. PIN.H

```
#ifndef PIN_H
#define PIN_H

#include <QPushButton>

//! [0]
class Pin : public QPushButton
{
    Q_OBJECT

public:
    Pin(int digit, QWidget *parent = 0);

signals:
    void clicked(int digit);

private slots:
    void reemitClicked();

private:
    int myDigit;

};
//! [0]

#endif
```

A.1.5. PIN.CPP

```
#include <QtGui>

#include "pin.h"

#include <QPushButton>

Pin::Pin(int digit, QWidget *parent)
    : QPushButton(parent)
{
    myDigit = digit+22;
    setText(QString::number(myDigit));
    connect(this, SIGNAL(clicked()), this,
SLOT(reemitClicked()));
}

void Pin::reemitClicked()
{
    emit clicked(myDigit);
}
```

A 1.6. PWM.H

```
#ifndef PWM_H
#define PWM_H

#include <QSpinBox>

class Pwm : public QSpinBox
{
    Q_OBJECT

public:
    Pwm(int numero, QWidget *parent = 0);

signals:
    void valueChanged(int value);
    void emitir_pwm(int value, int contador);

private slots:
    void remitpwm(int value);
private:
    int numero_pwm;
    int ciclo;

};
//! [0]

#endif
```

A1.7. PWM.CPP

```
#include <QtGui>

#include "pwm.h"

#include <QSpinBox>
#include <ros/ros.h>

Pwm::Pwm(int numero, QWidget *parent)
    : QSpinBox(parent)
{

    numero_pwm=numero;
    connect(this, SIGNAL(valueChanged(int)), this,
        SLOT(remitpwm(int)));

}

void Pwm::remitpwm(int value)
{
    ciclo=value;
    emit emitir_pwm(ciclo,numero_pwm);
}
```


}

A1.8. NODO.H

```
#ifndef NODO_H
#define NODO_H

/*****
****
** Includes
****
*****/
#include <std_msgs/Int32.h>
#include <ros/ros.h>
#include <string>
#include <QThread>
#include <QStringListModel>
#include <QObject>
#include <std_msgs/Int16.h>
#include "interfaz_arduino/adc.h"
#include <std_msgs/MultiArrayLayout.h>
#include <std_msgs/MultiArrayDimension.h>
#include <std_msgs/Int16MultiArray.h>

class Nodo : public QThread {
    Q_OBJECT

public:
    Nodo(int argc, char *argv[]);

    void run();
    void iniciar();
    void digital(const std_msgs::Int16& cmd_msg);
    void analogica(const interfaz_arduino::adc& adc_msg);

signals:

    void cambiar_pin(int i);
    void cambiar_analogico(float v0,float v1,float v3,float v4,float
v5,float v6,float v7,float v8,float v9,float v10,float v11,float
v12,float v13,float v14,float v15);

private:
    int init_argc;
    char** init_argv;
    ros::Subscriber digital_s;
    ros::Subscriber analog_s;
};

#endif /* interfaz_QNODE_HPP_ */
```

A1.9. NODO.CPP

```
#include <ros/ros.h>
#include <ros/network.h>
#include <string>
#include <std_msgs/String.h>
#include <sstream>
#include "nodo.h"
#include <std_msgs/Int32.h>
#include <ros/ros.h>
#include <std_msgs/Int16.h>
#include "interfaz_arduino/adc.h"
#include <std_msgs/MultiArrayLayout.h>
#include <std_msgs/MultiArrayDimension.h>
#include <std_msgs/Int16MultiArray.h>

Nodo::Nodo(int argc, char *argv[]) {
ros::init(argc, argv, "arduino");
interfaz_arduino::adc adc_msg;
}

void Nodo::run() {
ros::spin();
}

void Nodo::iniciar() {
ros::start(); // Necesario con el nodo fuerad del scope.
ros::NodeHandle n;
digital_s = n.subscribe("lectura_digital", 1000, &Nodo::digital,
this);
analog_s=n.subscribe("lectura_analogica", 1000, &Nodo::analogica,
this);
start();
}

void Nodo::digital(const std_msgs::Int16& cmd_msg) {

int i=cmd_msg.data;
emit cambiar_pin(i);
}

void Nodo::analogica(const interfaz_arduino::adc& adc_msg) {
float v0,v1,v2,v3,v4,v5,v6,v7,v8,v9,v10,v11,v12,v13,v14,v15;

v0=adc_msg.ADC0/204.6;
v1=adc_msg.ADC1/204.6;
v2=adc_msg.ADC2/204.6;
v3=adc_msg.ADC3/204.6;
v4=adc_msg.ADC4/204.6;
v5=adc_msg.ADC5/204.6;
v6=adc_msg.ADC6/204.6;
v7=adc_msg.ADC7/204.6;
v8=adc_msg.ADC8/204.6;
v9=adc_msg.ADC9/204.6;
v10=adc_msg.ADC10/204.6;
```



```
v11=adc_msg.ADC11/204.6;  
  
v12=adc_msg.ADC12/204.6;  
v13=adc_msg.ADC13/204.6;  
v14=adc_msg.ADC14/204.6;  
v15=adc_msg.ADC15/204.6;  
emit  
cambiar_analogico(v0,v1,v2,v3,v4,v5,v6,v7,v8,v9,v10,v11,v12,v13,v14  
,v15);  
  
}
```

ANEXO 2 arduino/ros

```
#include <WProgram.h>
#include <ros.h>
#include <std_msgs/Int8.h>

#include <interfaz_arduino/adc.h>
#include <interfaz_arduino/pwm.h>
#include <std_msgs/Int16.h>
#include <std_msgs/MultiArrayLayout.h>
#include <std_msgs/MultiArrayDimension.h>
#include <std_msgs/Int16MultiArray.h>

ros::NodeHandle nh;
interfaz_arduino::pwm pwm_msg;
interfaz_arduino::adc adc_msg;
std_msgs::Int16 digital_msg;
std_msgs::Int16 valor_msg;
std_msgs::Int16MultiArray array;

int averageAnalog(int pin){
    int v=0;
    for(int i=0; i<4; i++) v+= analogRead(pin);
    return v/4;
}

void pwm(const interfaz_arduino::pwm& pwm_msg){

    analogWrite(2,pwm_msg.pwm2);

}

void configuracion( const std_msgs::Int16& cmd_msg){
    int aux=1;
    for(int i=22; i<38;i++){
        if((cmd_msg.data & aux)==aux){

            digitalWrite(i,HIGH);
        }
        else{
            digitalWrite(i,LOW);
        }
        aux=aux*2;
    }
}

ros::Subscriber<interfaz_arduino::pwm> sub2("pwm", &pwm);
ros::Subscriber<std_msgs::Int16> sub("entradas_digitales",
&configuracion );
```



```
ros::Publisher digital("lectura_digital",&digital_msg);

ros::Publisher analogico("lectura_analogica", &adc_msg);

void setup()
{
    for(int i=2;i<13;i++){
        pinMode(i,OUTPUT);
    }

    for(int i=22; i<38;i++){
        pinMode(i, OUTPUT);
    }
    for(int i=38; i<53; i++){
        pinMode(i, INPUT);
    }
    nh.initNode();
    nh.subscribe(sub);
    nh.subscribe(sub2);
    nh.advertise(digital);
    nh.advertise(analogico);
}

void loop()
{
    int valor=0;
    int j;
    int n=0;
    byte dato=0;
    for(j=38;j<53;j++)
    {

        valor=digitalRead(j);
        dato=dato|(valor<<n);
        n++;
    }
    digital_msg.data=dato;
    digital.publish(&digital_msg);

    adc_msg.ADC0 = analogRead(A0);
    adc_msg.ADC1 = analogRead(A1);
    adc_msg.ADC2 = analogRead(A2);
    adc_msg.ADC3 = analogRead(A3);
    adc_msg.ADC4 = analogRead(A4);
    adc_msg.ADC5 = analogRead(A5);
    adc_msg.ADC6 = analogRead(A6);
    adc_msg.ADC7 = analogRead(A7);
    adc_msg.ADC8 = analogRead(A8);
    adc_msg.ADC9 = analogRead(A9);
    adc_msg.ADC10 =analogRead(A10);
    adc_msg.ADC11 = analogRead(A11);
    adc_msg.ADC12 = analogRead(A12);
    adc_msg.ADC13 = analogRead(A13);
    adc_msg.ADC14 = analogRead(A14);
    adc_msg.ADC15 = analogRead(A15);

    analogico.publish(&adc_msg);
}
```

```
nh.spinOnce();  
  delay(1);  
}
```